Abstract

Williams, Scott Everett. The Lageos Satellite: A Comprehensive Spin Model and Analysis. (Under the direction of Dr. Arkady Kheyfets).

A thorough investigation into the theoretical modeling of the Laser-Ranged Geodynamics Satellite (Lageos I) spin state evolution is presented. Starting from an existing dynamical model, we analyze in detail each of the model's assumptions and explore possible enhancements. Additional concerns not considered by the original model are also scrutinized in a bottom-up approach. In particular, we re-evaluate the orbit propagation module, survey and investigate all possible space-environment effects, assess numerical implementation concerns, and perform a number of software feature modifications. In the process, a parameterized approach is adopted and corresponding non-linear optimization tools are integrated into the revamped model. The outcome is a comprehensive, open-source model of the Lageos I spin dynamics which exhibits a significant advance in predictive accuracy. A corollary of the effort is a broad survey of the important space environment effects on the attitude of passive satellites.

In addition, a thorough analysis of the model results is presented along with an expanded discussion of the interesting discoveries we made. Particularly significant is

Report Documentation Page

Form Approved
OMB No. 0704-0188

1. REPORT DATE	2. REPORT TYPE	3. DATES COVERED
00 DEC 2002	**N/A**	**-**

4. TITLE AND SUBTITLE	5a. CONTRACT NUMBER
The Lageos Satellite: A Comprehensive Spin Model And Analysis	5b. GRANT NUMBER
	5c. PROGRAM ELEMENT NUMBER
6. AUTHOR(S)	5d. PROJECT NUMBER
	5e. TASK NUMBER
	5f. WORK UNIT NUMBER
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) **North Carolina State University Raleigh, North Carolina**	8. PERFORMING ORGANIZATION REPORT NUMBER
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)	10. SPONSOR/MONITOR'S ACRONYM(S)
	11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release, distribution unlimited

13. SUPPLEMENTARY NOTES

14. ABSTRACT

15. SUBJECT TERMS

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			
unclassified	**unclassified**	**unclassified**	**UU**	**266**	

Standard Form 298 (Rev. 8-98)
Prescribed by ANSI Std Z39-18

the sensitivity of the spin state evolution to small changes in the principal moments of the satellite—an idea discounted by previous efforts that nevertheless can be analytically verified.

A consequence of the effort is the immediate application to a number of ongoing research activities involving the Lageos I satellite. Of particular interest is the potential role of Lageos I in a proposed experiment to measure the general relativistic force known as gravitomagnetism. A precise understanding of the evolution of Lageos' spin dynamics is required so that correlated thermal effects may be properly accounted for in the evaluation of orbital motion. A related effort is the attempt to empirically measure the spin state based on optical glint data. This process must be seeded with a quality initial estimate of the spin axis orientation for proper evaluation of the data. The model we present has implications for both of these efforts.

THE LAGEOS SATELLITE: A COMPREHENSIVE SPIN MODEL AND ANALYSIS

by

SCOTT EVERETT WILLIAMS

A dissertation submitted to the Graduate Faculty of
North Carolina State University
in partial fulfillment of the
requirements for the Degree of
Doctor of Philosophy

APPLIED MATHEMATICS

Raleigh, North Carolina
December 2002

APPROVED BY:

A. Kheyfets Chair of Advisory Committee	R.O. Fulp
P.A. Gremaud	L.K. Norris

Dedication

For my wife, Tara. Your loving support, your faith, and your encouragement has sustained me these past months.

For my girls, Madison and Bailey, your welcome 'interruptions' were often just what I needed to reenergize and refocus. Thank you for being "Daddy's little girls."

For the countless army of friends and family who have held me in your prayers during this work. It could not have been done without your support.

And, for my Lord and Savior Jesus Christ. Apart from Him all this would be meaningless. Philippians 4:13

Biography

Scott Everett Williams was born December 4, 1969 in St. Paul, Minnesota. Shortly thereafter, Scott's family moved to Oregon where he grew up. He graduated with honors from Lakeridge High School near Portland in 1987.

Scott then headed to the East Coast where he earned a Bachelors of Science in Mathematics from Massachusetts Institute of Technology in 1991. He attended MIT courtesy of an Air Force ROTC scholarship and was commissioned a Second Lieutenant upon graduation. During his undergraduate years, he also served as captain for the baseball team, was a member of the Lambda Chi Alpha fraternity, and led a number of weekly bible studies.

A month after graduating from MIT, Scott married his wife Tara and moved to Tacoma, WA. While Scott waited for his first Air Force assignment, he developed curriculum for the Tacoma public schools' Math, Engineering and Science Achievement (MESA) classes.

In 1992, Scott and Tara moved to Los Angeles, CA where Scott donned his Air Force uniform for the first time. While there, he managed a mission software development contract for an early warning satellite constellation, and later, was the operational lead for the launch and deployment of a defense communication satellite. He also was selected to

participate in an engineering exchange assignment with the Aerospace Corporation where he wrote software for astrodynamics applications.

In 1996, the Math Department at the United States Air Force Academy in Colorado Springs, CO chose Scott for a teaching position. The assignment included a sponsorship for a Master of Science degree in Applied Mathematics, which Scott earned from North Carolina State University in 1998. Back at the Academy, Scott taught courses in both differential and multivariable calculus and assisted with the development of instructional software. During their stay in Colorado, Scott and Tara celebrated the birth of their first daughter, Madison Rebecca.

Scott returned to Raleigh in 1999 to pursue a Ph.D. in Applied Mathematics, sponsored once again by the Air Force. In addition to academic rigors, he has taken on various leadership positions at his church, including small group leader, retreat speaker, and men's ministry coordinator. Scott and Tara had another beautiful little girl, Bailey Elaine, in 2001.

Scott currently serves as a Major in the United States Air Force and is headed for an assignment at Ramstein Air Force Base in Germany.

Acknowledgments

A work of this magnitude is challenging regardless of circumstance. However, for me (and my family), the intense final months of this effort came amidst a number of life-disrupting situations. It is only in that larger context that I can begin to acknowledge the support that has made this work possible.

Foremost, is my advisor Dr. Arkady Kheyfets; confident, flexible, and always positive. I thank you for guiding me through this process and making it a rewarding venture for me. I hope it has been rewarding for you as well.

To my committee members—Dr. Fulp, Dr. Gremaud, and Dr. Norris—I am grateful to have had you on my 'team.' You have been helpful in your advice, generous with your time, and accommodating with your schedules. Thank you.

For my wife Tara. She may not have written any code or solved any equations, but she worked every bit as hard as I did over the course of this project, and so, it is hers as much as it is mine. It was she who kept our family functioning while I labored, often doing the work of two parents amidst unsettling personal circumstances. Through it all she remained my biggest encourager. Words cannot express my gratitude, appreciation, and love.

A legion of friends and family have had an immeasurable impact in more ways than can be imagined. You made space in your home when we were displaced from ours, toiled with us to meet our needs, and, above all else, held us close in your prayers. Without that support, this project would never have been completed. This work is the fruit of your labor as well; I hope it is a project worthy of your efforts.

While I am loath to single anyone out from all who have supported us, I must say a special thanks to Bill and Kymberly Arana and my father, Tim, for taking time to read my manuscript. The work is greatly improved because of your efforts; thank you for your sacrifice on my behalf.

To my Pastor Bill Gross, my teacher, mentor, and friend, there was far more to my education here than just academics. Thank you for your guidance, encouragement, and confidence.

I am also indebted to my friends and colleagues in the Department of Mathematics at the United States Air Force Academy, and Col Litwhiler in particular. I would not be here at all if not for your confidence in my abilities. Thank you for the opportunity; I look forward to putting this degree to use when I return in a few years!

Above all, I wish to offer all praise and glory to my Lord Jesus Christ. In times when the magnitude of the effort threatened to overwhelm, You reminded me of the proper context of it all. I can only echo the words of Paul, "I consider everything a loss compared to the surpassing greatness of knowing Christ Jesus my Lord." Amen.

Table of Contents

List of Tables

List of Figures

1 Introduction and Historical Context

1.1 Overview

We set out to explore a problem that has been solved many times over and yet has never really been solved at all. In this case, our motivation is not abstract understanding but rather specific application. Namely, we seek a quantitatively accurate model of the spin dynamics of the Lageos I satellite.[1]

As perhaps the most precisely tracked of any artificial satellite [D], Lageos is an excellent instrument for detecting small and heretofore undetected orbit perturbing forces. Of particular interest in this regard is Lageos' role in a proposed experiment to measure the general relativistic force known as gravitomagnetism [Ciufolini, 1986]. To succeed in these efforts, however, a precise understanding of the evolution of Lageos' spin state is required so that correlated thermal effects can be properly accounted for in the evaluation of orbital motion.

---

[1] A second Lageos type satellite (II) is also on orbit and a third, as will be discussed, has been proposed. However, our focus throughout is on Lageos I, and so it will be convenient hereafter to drop the "I" identifier unless context demands otherwise.

Accordingly, various attempts have been made to model the Lageos spin dynamics. Unfortunately, however, these efforts have met with more qualitative than quantitative success. While qualitative results are useful in the general discussion of the problem, a quantitative (i.e., predictive) model is necessary if spin-related orbit perturbations are to be sufficiently addressed in the experiment. With our current effort, we are able to show a significant improvement over the previous efforts in predicting the spin state of the Lageos satellite. These ideas are expounded in the sequel.

The statement of the problem is simple enough,

$$\frac{d\boldsymbol{L}}{dt} + \boldsymbol{\omega} \times \boldsymbol{L} = \boldsymbol{N} \qquad 1$$

where $\boldsymbol{L}$ is the spin angular momentum[2], $\boldsymbol{\omega}$ is the spin angular velocity, and $\boldsymbol{N}$ is the torque due to external and/or internal influences (see e.g., Goldstein [1980]). Equation 1, known as Euler's equations of motion, is a fundamental topic in any first course on rigid body mechanics. Indeed, it is interesting that a problem so old and basic in expression continues to confound in so many ways. Avizonis [1997] remarked on the deceptive simplicity of the system, which upon closer inspection is anything but simple.

The primary product of our efforts is an open source computer model of the Lageos spin dynamics. Accordingly, significant attention is devoted to issues related to the

---

[2] It should be noted that Euler's equations of motion are general. We make the restriction to "spin" only because that is the present concern. Moreover, to first order, rotational dynamics on different scales such as a satellite's orbital and body spin motions are not coupled and may be treated separately. In fact, there is coupling at higher orders and that is in part what motivates the current interest in Lageos' spin dynamics.

numerical implementation and software development. This is done for two reasons. First, in a rush to get results, important and frequently non-trivial numerical issues are often overlooked; it is our goal to avoid this trap. Second, we wish to provide future users of the model with an understanding of the code and inform of the decisions and lessons learned along the way. We do so with the anticipation of aiding future model adaptations to specific applications.

The model itself does not originate with us but rather was first introduced by Habib et al, [1994]. It has since undergone numerous revisions (including the present author's efforts pre-dating this report [Williams, 1997]). This current effort is not a culmination, but it does provide substantial refinement in the evolution of the model. There remains much that could be done to further improve upon the results we achieved. These possible refinements are identified and discussed in our conclusion.

Consistent with this viewpoint, we engage topics throughout either by identifying deficiencies with the existing model or by observing opportunities to enhance the fidelity and reach of the model. As such, the underlying mechanics are approached as a means to an end (implementation), rather than as an end themselves.

The empirically determined spin state data of Avizonis [1997] is used as a benchmark for comparison with our own results. An example comparing Avizonis' data with earlier model output is shown in Figure 1.1. It is interesting to note that there are present efforts underway to apply Avizonis' approach to more recently recorded data [Currie, private communication]. However, the current real-world dynamics make spin state solutions

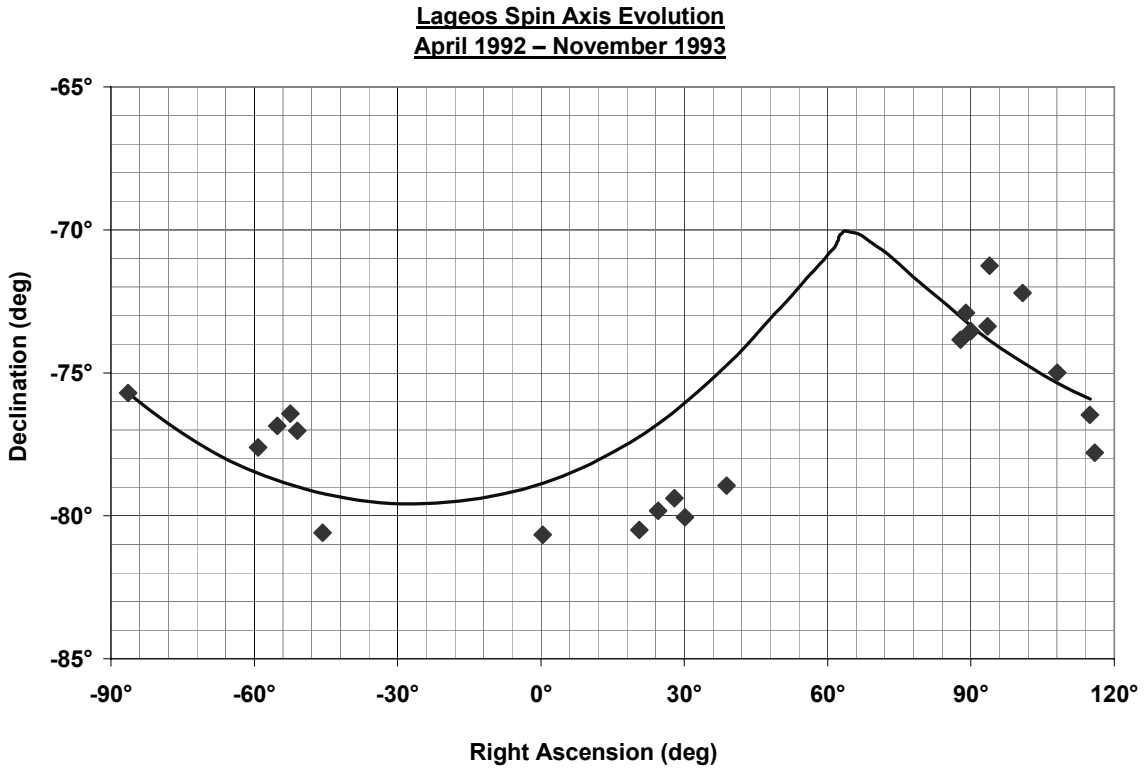

Figure 1.1 Comparison of theoretical and empirical Lageos spin axis evolution data

Empirically measured Lageos I spin axis orientation as determined by Avizonis [1997] during the period April 1992 to November 1993. A sample output from an earlier version of our model is also shown for reference (solid line). The spatial proximity of the model curve to Avizonis' solutions shows good qualitative agreement between model and the empirical data but space-time correlation (not shown) was relatively poor. The latest model performs much better in this area.

impossible without seeding the process with a quality *a-priori* estimate of the orientation. In this regard, our work has an immediate application toward that effort. This is detailed later in our discussion.

A consequence of our efforts is a significant collection of technical, structural, and dynamical information about the Lageos satellite. We convey much of that information

here to provide an encyclopedic resource. However, some 'original source' documents were difficult to obtain, and so secondary (and sometimes conflicting) sources were used.

Supporting material derives from several sources that merit distinct citation. Traditional published references are given by author and publication year (e.g., Williams [1997] or [Williams, 1997]); the bibliography is alphabetized. Additionally, a great deal of information was gleaned from the Internet. Website citations are listed separately from the publication bibliography and are identified by a capital letter relating to the citation entry (e.g., [H]). To the best of our knowledge, the URLs provided are current and accurate; however, due to their transient nature, they may change over time. Finally, several different numerical packages have been accessed. Because the sources and accompanied documentation vary, these are referenced in a third list and identified by Roman Numerals (e.g., [ii]).

Chapters 2 and 3 respectively contain a thorough treatment and development of the Equations of Motion and the related torques specific to the Lageos satellite. Also found in Chapter 2 is an overview of conventions and definitions as well as a comprehensive review of the Lageos satellite's physical and dynamical properties. Chapter 3 details all aspects of the modeling effort from the torques already mentioned to software and numerical concerns. Finally, in Chapter 4 we present some of the interesting results of this effort and discuss implications for future work. The remainder of this section is devoted to placing our work in its larger context and discussing other related contributions.

1.2 Synopsis of the Lageos I Satellite

While we defer detailed discussion about Lageos to Chapter 2, some background on the satellite and its mission is necessary to appreciate the ongoing interest in its dynamics. The name, LAGEOS, is an acronym for **LA**ser-Ranged **GEO**dynamic **S**atellite. As the name implies, Lageos is a laser-tracked satellite whose position can be determined with great accuracy. This allows for high precision measurements of geodynamical phenomena: tectonic plate motion, gravity field gradient, nutation of the Earth spin axis, etc. [A]

The satellite itself is a simple geometric structure—two hollowed out hemispheres of aluminum encasing a cylindrical core of Beryllium Copper and bolted together along a tension stud. The surface of the hemispheres is covered with cube corner retroreflectors to reflect ground station initiated laser tracking signals. A picture of the satellite is shown in Figure 1.2 (page 7).

Lageos travels in a highly inclined retrograde and nearly circular orbit at an altitude of about one Earth Radii. It completes a little over six and a third orbit revolutions per day. The satellite was designed for, and the specific orbit selected to minimize potential orbit perturbing influences [D].

Of particular interest is the precession of Lageos' orbit plane with respect to an Earth-centered inertial coordinate system (designated ECI and specifically defined in the sequel). Indeed, this motion is a 'macroscopic' (from the viewpoint of spin dynamics)

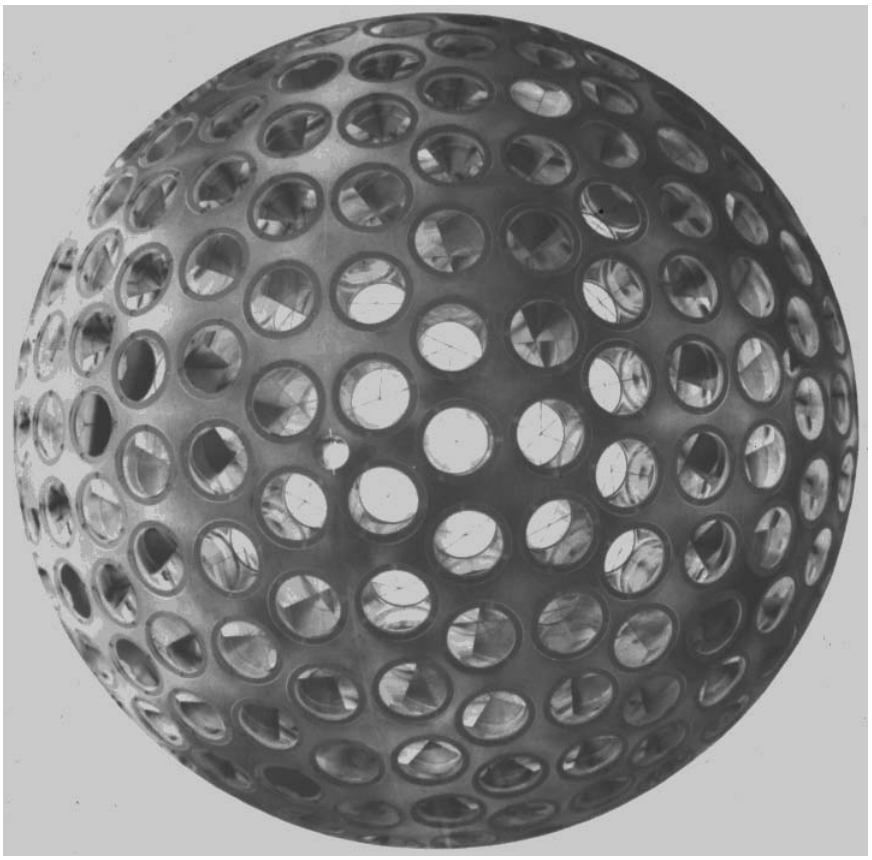

Figure 1.2 Picture of the Lageos I satellite.

case of (1).[3] Because the orbit is inclined with respect to the equator, the motion is subject to gravity gradient torques resulting from the Earth's non-uniform mass distribution. This causes the orbit plane to precess. Higher order effects, both gravitational and non-gravitational, also affect this motion. All together, the Lageos satellite experiences a net precession period of about 34 and ½ months [F].

[3] That is, "equation 1." We use this direct reference style for equations throughout the discussion.

As a passive satellite, i.e., no mechanisms for attitude control, Lageos' subsequent motion (spin and orbit) is completely determined by its interaction with the space environment. To initially mitigate some of the aforementioned affects, Lageos was "spun-up" along its axis of symmetry to confer energy and a stable orientation at the time of orbit insertion. Over time, however, interaction between the Earth's magnetic field and the satellite's metallic body give rise to energy dissipating eddy currents, allowing for increasing susceptibility to secondary environmental torques.

1.3 The Case for Spin State Determination

1.3.1 Lageos & Small Orbit Perturbations

Over the years, the higher order effects on Lageos' precessional motion have generated considerable interest for study of issues unrelated to its original mission. For most orbiting satellites, position determination "noise" conceals small orbit perturbing effects. In particular, the error associated with predicting the precession caused by the gravity-gradient dwarfs the microscopic contributions from other forces. For Lageos, however, orbit position determination to the centimeter level makes previously masked effects observable and measurable. Reflexively, this demonstrates the need to include these

forces in the dynamical models aimed at generating orbit position data [e.g., Farinella & Vokrouhlicky, 1996].[4]

It is remarkable that a satellite exceedingly unsophisticated by comparison to many of today's technically advanced spacecraft, is nevertheless the focus of so much attention after nearly 30 years on orbit. Lageos' elegant simplicity allows for precise observational measurements and makes accurate theoretical modeling possible. The point of intersection of the two approaches is yielding new and better understandings of previously ill-defined concepts. For example, Rubincam [1990] has remarked,

> "The theory of charged particle drag was in sad shape for many years . . . However, the theory of Al'pert gives good agreement with what is observed on LAGEOS . . . So it looks like we finally have closure between theory and observation in the realm of charged particle drag."

Clearly, the evaluation of the secondary orbit-disturbing forces are intriguing problems in their own respect. Analysis involving Lageos data has improved knowledge about charged particle drag, solar radiation pressure including eclipse responses, and various radiation effects due to the Earth (optical and infrared). Lageos data has also contributed to refinements in models of the Earth's gravity field. Iorio [2000] provides a comprehensive evaluation of all these effects.

---

[4] In fact, this is typically an iterative process. Results generated by dynamical models are compared with observations; differences exceeding predicted errors are noted and plausible sources are explored and tested. The process is repeated until consistent results are obtained. This can, and has, led to disagreement over root causes of the effects. See e.g., Rubincam [1990].

1.3.2 Spin State Dependent Effects

Of particular interest are perturbations leading to a secular decay in the semi-major axis of Lageos' orbit. Among the contributing factors, the dominant force is Yarkovsky thermal drag,[5] which accounts for approximately 70% of the decay. Yarkovsky drag is a type of recoil force that results from the anisotropic heating and cooling of the satellite's surface via radiative sources [Rubincam, 1990]. Distinctions are sometimes made between seasonal (orbit period), daily (spin period), and eclipse-related type Yarkovsky, but all such forces depend on the spin state of the satellite.

To understand the forces at work, consider a rotating body in the presence of a radiation source (e.g., satellite orbiting the Earth). Radiation from the Earth heats the direct facing satellite surface. As the body rotates, the satellite surface spins away from the heat source and re-radiates the absorbed energy to space. There is a lag between the process' heating and cooling due to thermal inertia. Consequently, the momentum of the particles leaving the satellite during cooling is in a different direction than the momentum received during heating. This results in a net force, a component of which acts to oppose the orbital motion. The force is dependent both on the spin rate and on the spin axis orientation.

For this reason alone, there has been great interest in determining the spin state of the Lageos satellites. The implications are cascading—better spin state knowledge allows

---

[5] The effect goes by several names in the literature including "Radiation Rocket" [Bertotti & Iess, 1991], "Rubincam effect" [Iorio, 2000], "thermal thrust" [Farinella & Vokrouhlicky, 1996].

better modeling of the Yarkovsky affect; in turn, this improves satellite position determination, which leads to even more precise measurements of geodynamic phenomena.

There is also a kind of reverse cascading implication. Many forces are smaller still than Yarkovsky drag, yet they contribute non-trivially to the perturbed motion of the satellite. Unfortunately, it is difficult, if not impossible to identify and separate these forces from the data if the Yarkovsky effect is not modeled with sufficient accuracy. One of these forces in particular has captured a great deal of interest and motivates much of the previously mentioned efforts; gravitomagnetism.

1.3.3 Gravitomagnetism and the Lense-Thirring Clock Effect

According to classical Newtonian mechanics, the only gravitational force is the familiar $-1/r^2$ type produced by the mass of a body. In his theory of General Relativity, however, Einstein posited an additional gravitational force based not only on the mass itself but also on the motion of the mass, or "mass currents." This yields a $-1/r^3$ effect akin to that of a magnetic field, hence the name gravitomagnetism.

To illustrate, we borrow an example from Habib et al [1994]. Place a satellite in a polar orbit about an idealized Earth-like mass, not spinning with respect to distant space. The orbit plane will remain fixed in its orientation in space. Now spin the central mass. The orbit plane of the satellite will experience torque along the central body's rotation axis causing a precession of the orbit plane because the rotating mass has generated a dipole gravitational field, i.e., a gravitomagnetic field. This gravitomagnetic-induced

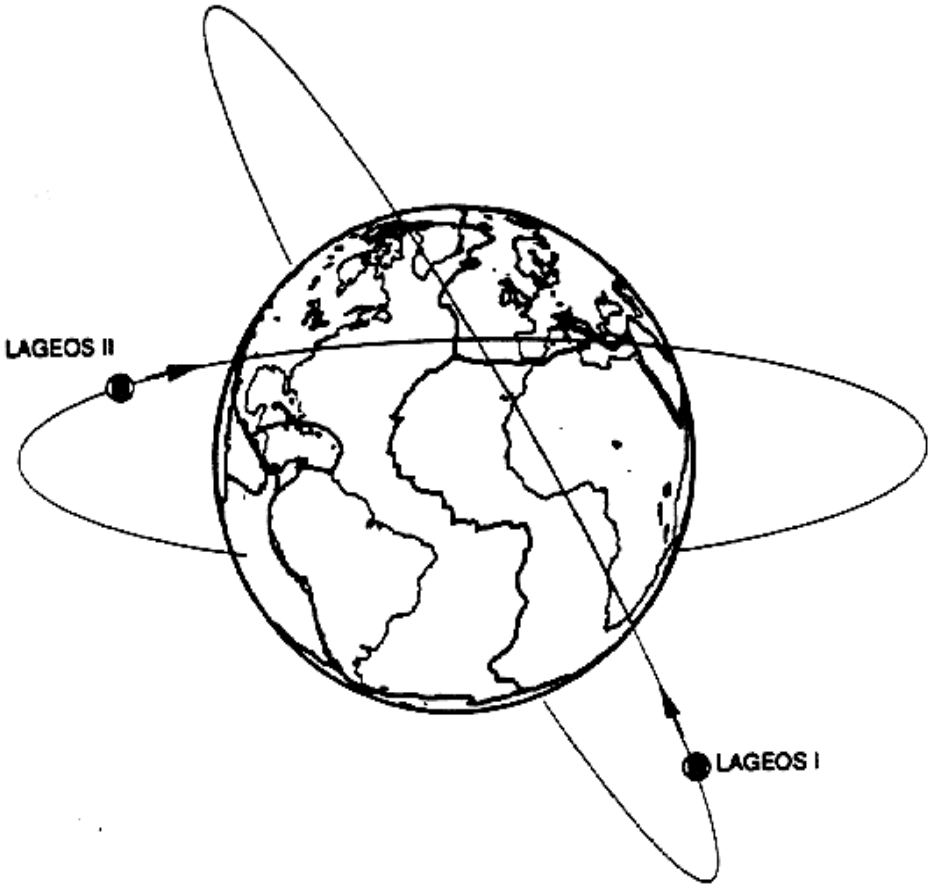

Figure 1.3 Orbit orientations for Lageos I and Lageos II

precession is known as the Lense-Thirring effect; although it is sometimes also referred to as frame-dragging or the gravitomagnetic clock effect. Whatever the name, for Lageos the result is a orbital precession of 32 milliarcsec per year.

The importance of measuring this force goes beyond the task of providing minute corrections to satellite orbit predictions. It is also, as Ciufolini & Wheeler [1995] have remarked, an important direct test on the theory of general relativity. Moreover, while

nearly negligible on the scale of Earth-bound satellites, gravitomagnetism is believed to play a significant role in large-scale astrophysical phenomenon.

To date, the effects of this force have not been completely verified, though Ciufolini [2002] has now come close. He achieved a measurement within 20% of theoretically predicted values by using a combination of data from the Lageos I and Lageos II satellites. Among the factors limiting the quality of the result is the previously identified uncertainty in orbit propagation due to effects related to the spin of the Lageos satellite. There is also a difficulty with the non-complementary relationship of the two orbits (Lageos II is prograde but has a significantly lower inclination than does Lageos I—see Figure 1.3), which is sub-optimal for reasons discussed in the following section.

1.3.4 The Lageos III Experiment

Notwithstanding the difficulties with Lageos I and Lageos II above, it is still widely believed that the Lageos satellites remain the best candidates for improving upon current empirical estimates to the Lense-Thirring effect. One of the most promising approaches, due to Ciufolini [1986], is the proposal to launch a third Lageos type satellite, Lageos III, in an orbit complimentary to Lageos I (i.e., identical orbital elements with a complimentary inclination). This provides what Habib et al [1994] refer to as a "tandem generated gyro plane" resulting from the equal but opposite (classical) precession of the two individual orbits. The large-scale effects, specifically those generated by the gravitational zonal harmonics of the Earth, cancel so that only higher order perturbations remain in the motion of the tandem plane.

And so we return to our earlier discussion! Neglecting the small error from imperfect cancellation of the classical forces, the remaining contributions to the motion of the tandem plane must be isolated. Iorio [2000] details the various effects and concludes that the Rubincam (i.e. Yarkovsky) effect is the dominant error source. So too, numerous other authors acknowledge that the success of the Lageos III experiment hinges on a successful determination throughout of the spin state of the two satellites (e.g., Rubincam [1990], Habib et al [1994], Farinella & Lucchesi [1991], Bertotti & Iess [1991]).

1.4 Lageos Spin Axis Modeling and Prediction

Two basic approaches have been employed to determine the spin state evolution of the Lageos satellite. The first is empirical—making spin state determinations from observed data. The second, the approach we have taken, is to employ a theoretical model of the dynamics based on the underlying mechanics.

1.4.1 Empirical Studies & Avizonis' Method

Initial efforts at empirical solutions attempted to 'back out' spin state information from orbital data. That is, orbit data is compared against model predictions of residual effects due to the spin-dependent thermal forces. A spin state evolution is then postulated and inserted into the model based on the evidence, and the process is repeated until a best fit is achieved. Farinella & Vokrouhlicky [1996] summarize several such efforts including

the original analysis by Rubincam [1987, 1990] in which he also predicted a spin state orientation at launch.

Avizonis - 1997

More recently, and perhaps more promising, Avizonis [1997] employed a more direct method based on optical glint telemetry. Because of the relatively simple geometry of Lageos, it is possible to determine the angle of the reflection surface for optical 'flashes' from the sun. Using a 'burst' of several such flashes, the latitude band of the reflectors involved can be determined (based on the flash frequency) thereby deriving the satellite's orientation and spin frequency. For a slowly evolving spin orientation, multiple 'bursts' in proximity (corresponding to different latitude bands due to orbital motion) can be used to refine and/or give confidence to the solutions obtained.

This represents substantial improvement over the previous empirical efforts in which results were implicit and perhaps model dependent. Still, Avizonis' approach has some limitations. For one, it works best at higher spin rates because multiple 'flashes' can be observed during a single 'burst.' Without multiple flashes, identifying the latitude band of the reflectors is much more difficult. Second, his method assumes that the body and spin axes are aligned. This is not alarming; almost every analysis to date makes the same assumption, as do we. In fact, excepting an anomalous post-launch spin-up, the body and spin axes should be aligned. Nevertheless, some literature references put the issue in doubt (e.g., Barlier et al, [1996]). Moreover, spin and body axis decoupling is a certainty in future years due to the decay of angular momentum. Even if the condition currently

holds, it will not remain that way for much longer. Finally, the Avizonis approach is unable to distinguish between *spatially inverted* pole solutions (same spin axis orientation in space but with opposite spin).

Currie - 2002

Currie [2002], among others, is currently reviving efforts to apply Avizonis' approach to more recent optical data sets. Because Lageos' spin angular velocity has decayed substantially from data points analyzed by Avizonis, these new efforts face a more challenging task. In addition to difficulties related to a slower spin rate, there may be more important considerations. In particular, neither of the fundamental assumptions in Avizonis' work—spin and body axes aligned and a slowly evolving body orientation over a set of bursts—may still hold. The latter is particularly problematic as it essentially undermines the use of data from multiple bands to 'triangulate' a solution.

As a remedy, Currie proposes a predictor-corrector type approach in which Avizonis' method is seeded with a predicted spin state from a dynamical model (such as our own). It should then be possible to interpret the optical data, i.e., identify the appropriate latitude band of reflectors, and refine the estimate. This would perhaps proceed iteratively until good agreement is reached between the model and the translated optical data.

There are reasons this approach is attractive. Theoretical predictions are always more appealing when correlated with directly observed data. On the other hand, it is unlikely that meaningful results can be derived from the recent optical data sets without a-priori

information that initially gets close to the solution. The cooperative effort could be far better than the individual results of either approach and may make a difference in meeting the stringent error requirements of the Lageos III experiment.

1.4.2 Dynamical Spin Models

Modeling the spin dynamics of Lageos from a theoretical standpoint presents numerous challenges. The greater the precision/fidelity asked of the model, the more costly and consuming it is to implement. For this reason, modeling efforts generally seek to meet the minimum requirements for a given application, but no more. The original model of Habib et al [1994], the evolutionary predecessor of our own work, is a good example. It sought only qualitative results; and therefore, took a number of liberties to facilitate the implementation.

Similarly, the majority of efforts view the theoretical spin state evolution as an intermediate result useful primarily in the context of computing orbit perturbing thermal forces (e.g., Rubincam [1987], Farinella & Vokrouhlicky [1996]). While this is the established motivation for spin state determination, there is an advantage to viewing the dynamical spin state modeling as a stand-alone effort. In particular, we have not been tempted to gloss over or "average out" potential effects based on a perceived lack of impact to the larger problem.

While our approach has advantages, we do not discount the preceding work's value. There are compelling features to some of the more detailed efforts that we recommend for closer examination. Moreover, much could still be done to further relax the

assumptions about the physical system as represented in our own model and so more closely approach the true system. Finally, some of the features we added in greater detail have produced only trivial responses. While the exercise was fruitful in that these refinements may prove useful as the model evolves, it affirms that within the fidelity of the current work, certain elements of the physical system can be ignored or at least simplified to a great extent.

Bertotti & Iess - 1991

The following dynamical models are specifically highlighted. The Bertotti & Iess [1991] model represents the first serious attempt to deal with the Lageos spin dynamics in their own right. The Bertotti & Iess model assert two principal torques governing Lageos' spin state evolution: 1) gravity gradient across the satellite's body and 2) the dissipative torque resulting from currents induced by Lageos' interaction with the Earth's magnetic field. Along the way, three notable decisions were made. The first was to consider only a reduced set of dynamical equations of motion rather than the full expression of (1). The second, which justifies the first, was an implicit assumption of relatively high spin rates[6]. Third, the angular velocity vector was assumed coincident with the body symmetry axis, allowing for a simplified expression of the magnetic torque.

---

[6] Bertotti & Iess analyze the low frequency case only after developing an approach with an implicit high frequency assumption. This led to interesting but not necessarily accurate behavior in the low frequency regime of the model.

Interestingly, while the general approach assumes a high spin frequency, Bertotti & Iess pursue a different rational to conclude that the low frequency limiting forms of the coefficients of magnetization (α' and α'') can be used from the beginning of the satellite's life. This is reasonable because of the way the true structure of the satellite 'violates' the simplifying assumptions made to compute the coefficients. To compensate, they introduce a scalar parameter to be determined post-priori based on observations. The effect is similar to our approach for the same issue—introduce parameters to compensate for inadequate representation of the physical system within the model.

Habib et al - 1994

Bertotti & Iess' work has been invaluable in modeling small orbit perturbations due to spin orientation. Still, some of their conclusions are questionable. In particular, they predict chaotic behavior for low rates of spin. Kheyfets [1992] and Habib et al [1994] refute this result, stating that the implicit assumption of high spin rates in the original derivations render the approach unsuitable for low frequency analysis. As a remedy, Habib et al defined a model based on the full set of dynamical equations (1) and a less restricted form of the coefficients of magnetization.

In pursuit of qualitative rather than quantitative results, the Habib et al model takes its own share of liberties with the physical system. They presume a magnetic dipole for the Earth aligned with the rotation axis. They also use a highly simplified orbit propagation, ignoring the precession of the orbit plane.

Habib et al show the spin axis motion remains reasonable throughout, and they identify three phases of motion: 1) a *Fast spin phase*, 2) a *Spin-orbit resonance phase*, and 3) an *asymptotic phase*. Briefly, for *fast-spin* the spin angular velocity is aligned along the body symmetry axis, and the total angular momentum experiences an exponential decay. As the spin period approaches the orbital period, the angular momentum transitions to an orientation orthogonal to the orbit plane, also decoupling from the body axis; this is the *spin-orbit resonance*. Finally, the *asymptotic phase* is characterized by a gradually settling of the dynamics and a "tidal locking" of the spin angular velocity to that of the orbit.

Barlier et al - 1996

In another work inspired by the Bertotti & Iess model, Barlier et al [1996] provide a more direct generalization of the original approach. Of particular interest are a refinement of the magnetic torque model, evaluation of cases with possible initial misalignment between the body spin and symmetry axes, and a claim that the initial (orbit insertion) pole solution for Lageos' spin-axis orientation by Rubincam [1990] may be spatially inverted.

The work retains Bertotti & Iess' use of a reduced set of motion equations based on the time-averaged approach, differing from our effort in that regard. Still, similarities persist, including efforts to treat some model elements as post-priori parameters to account for specifically unmodeled but notionally understood effects. Also, their mention of the possibility of a spatially inverted pole is consistent with our thinking

based on observed behavior from the present effort. This matter is discussed further in Chapter 4.

Williams - 1997

Finally, the present author was invited to revisit and generalize the work of Habib et al. In response, we [Williams, 1997] added a more realistic magnetic field model of the Earth—a dipole oriented to match the observed magnetic pole location at a co-latitude of about 11°. This required the insertion of numerous secondary features to account for the now time-dependent (due to Earth rotation) magnetic field. In addition to validating the general conclusions of the predecessor model, we also made strong initial progress toward a more predictive (quantitative) tool. The present effort represents a continuation of this process.

1.5 Summary

After nearly three decades on orbit, Lageos I remains an intriguing and important tool for geophysical research. Its uniquely precise orbit exhibits the effects of previously undetectable perturbing forces, including gravitomagnetism. To properly isolate these effects within the Lageos orbital data, a detailed understanding of surface thermal forces is required. In turn, this places an emphasis on evaluating the spin state of the satellite. To this end, numerous efforts have been made to model the Lageos spin dynamics, both empirically and theoretically. Unfortunately, these efforts fall short of the ideal and

cannot provide the predictive accuracy required for a sufficient determination of the thermal force effects. The present work aims to improve upon those results and we now proceed with the details.

2 Foundations

2.1 Conventions

In moving forward, it is useful to identify various conventions employed throughout this work. The following summary is presented to clarify the subsequent discussion.

Symbols and Notation

Whenever possible, we have used standard notations and definitions. However, to avoid confusion and to document decisions peculiar to this work, we establish the following:

- Scalar quantities are written in italics (x, σ), vectors and higher order counterparts as bold italic ($\boldsymbol{r}$, $\boldsymbol{\omega}$), and units in standard font (cm, s).
- The customary "hat" notation ($\hat{\boldsymbol{r}}$) is used for unit vectors, excepting the basis vectors discussed next.
- The principal directions for any Cartesian coordinate system are often referred to in the text as x, y, and z directions, but the corresponding basis vectors are identified as $\boldsymbol{e}_1$, $\boldsymbol{e}_2$, and $\boldsymbol{e}_3$ respectively; an arbitrary vector then has elements $\boldsymbol{r} = (r_1, r_2, r_3)$.

- Spherical coordinates are often referred to using "geographic" terminology—*longitude* for the *x*-*y* plane angle measured counter-clockwise from the *x*-axis; *co-latitude* for the polar angle measured from the *z*-axis; and *latitude* for the 90° complement of co-latitude, measured from the *x*-*y* plane.
- The dot notation is used for complete derivatives in the time variable: $\frac{d\varphi}{dt} \equiv \dot{\varphi}$.

Bookkeeping

Multiple coordinate systems are employed, making bookkeeping an issue. For instance, the usable form of the equations of motion is given in terms of variables relating one system of axes to another. In lieu of cascading layers of superscripts and subscripts, we shall often rely on the clarity of context. In particular,

- Reference system identifications are usually omitted for a vector occurring in its 'native' frame (e.g., $\boldsymbol{\omega}$ in the body system of axes) and with vectors for which no particular designation is yet required.
- For vectors in non-native frames, a superscript capital letter identifying the non-native frame is used (e.g., $\boldsymbol{\omega}^{\mathrm{L}}$).
- For generally defined vectors, such as $\boldsymbol{e}_i$, parentheses are used to indicate the original and resulting frames: e.g., $(\boldsymbol{e}_1^{\mathrm{E}})^{\mathrm{B}}$ is the B-frame representation of the E-frame *x*-axis unit vector.

Constants and Standards

The *cgs* (centimeters-grams-seconds) convention is used within the model, as opposed to the SI (Le Système International d'Unités, [I]) standard meters-kilograms-seconds. This is done because the cgs system is better suited for work involving electrodynamics (see Jackson [1975]), and it is simple enough to translate between the two systems.

The situation for physical constants is a little more complicated. The National Institute of Standards and Technology (NIST) maintains a list of "Fundamental Physical Constants" at [I]. However, this list contains few astrodynamical parameters, partly because most such quantities are not truly constant (thus "dynamic"). Moreover, many of the values important to this work are not direct measurements but are themselves parameters corresponding to the best fit of a particular model to observations of a physical system. For example, the specific values of the Geopotential coefficients (Chapter 3) are different for the JGM-3 model than for the EFM96S model (see Gill & Montenbruck [2000] for a historical summary of gravity models). Thus, while new and better values for a specific parameter may be achieved, legacy occasionally requires use of an older, less accurate set.[1]

---

[1] An example of this can be seen in the NORAD orbit model behind the Lageos orbit data we use (Section 0). The NORAD model uses the WGS-72 ("World Geodetic Survey – 1972") standard for geophysical parameters even though there are far better measurements available today (due in part to Lageos data). Adopting a new standard in this case requires reevaluating decades of orbital data (to ensure data sets for a given satellite remain internally consistent); a prohibitive task.

This constraint is true for the present effort. We draw from many sources (orbit data, geodesy, geomagnetic, etc.), and so several different standards are represented in the assembled model. While this is probably not much of an issue within the fidelity of the model, internal consistency is always desirable. For parameters within our control, we use values specified by the International Earth Rotation Service (IERS, [J]).[2]

Time

Much could be said about issues of time because it is intertwined with the astrodynamical system. Indeed, the very definition of time as we experience it, is based on the rotation of the Earth. However, in lieu of a protracted discussion, we refer to Kelso [1995-6] or Gill & Montenbruck [2000]. For our purposes, a few definitions suffice.

- *Mean Solar Day* is the period of the Earth's revolution with respect to the mean Sun location. By definition, it has a duration of 86,400 s.[3] This is the timescale of our everyday time keeping.
- The mean Sun drifts eastward by about 1 °/day due to the Earth's orbital motion; therefore, the Mean Solar Day is greater, by about 4 minutes, than the period of the Earth's revolution relative to fixed space, known as the *Mean Sidereal Day*.

---

[2] Some of the common reference standards include the WGS (-72, -84, -84/NIMA-97), the IERS, and the IAG; see Featherstone [1996] or [K] for additional information.

[3] Hours, minutes, and seconds were originally angular measures. One complete revolution has 24h*60m/h*60s/m = 86400 s and so begat the original definition of the length of a second—1/86400 of the duration of subsequent meridian transits of the Sun.

- *Universal time* (UT1) is a mean solar timescale that relates the angular displacement of the Greenwich Meridian from the Earth-Sun line to a clock-on-the-wall time (basis for time zones); 12h UT1 corresponds to Greenwich noon (0° angular displacement).
- *Julian Day* is a mean solar timescale expressed in day units with partial days as fractions. A Julian day starts and ends at 12h UT1.
- *Julian Date* (JD) is a monotonic Julian Day timescale for the *Gregorian Calendar* (the calendar we use in everyday life). E.g., January 1, 2000 at 12h UT1 =2451545.0 JD is the *J2000 Epoch*.
- *J2000 Julian Date* (JD2K) is a monotonic Julian Day timescale referenced to J2000. E.g., January 1, 2000 at 12h UT1 = 0.0 JD2K

2.2 Satellite Attitude Dynamics

To put (1) in usable form requires a brief review of rigid body kinematics. Using terminology directly from our application, we begin with a definition of the two fundamental Cartesian coordinate systems of the problem (Section 2.2.1) and discuss the relationship between the two (Section 2.2.2).

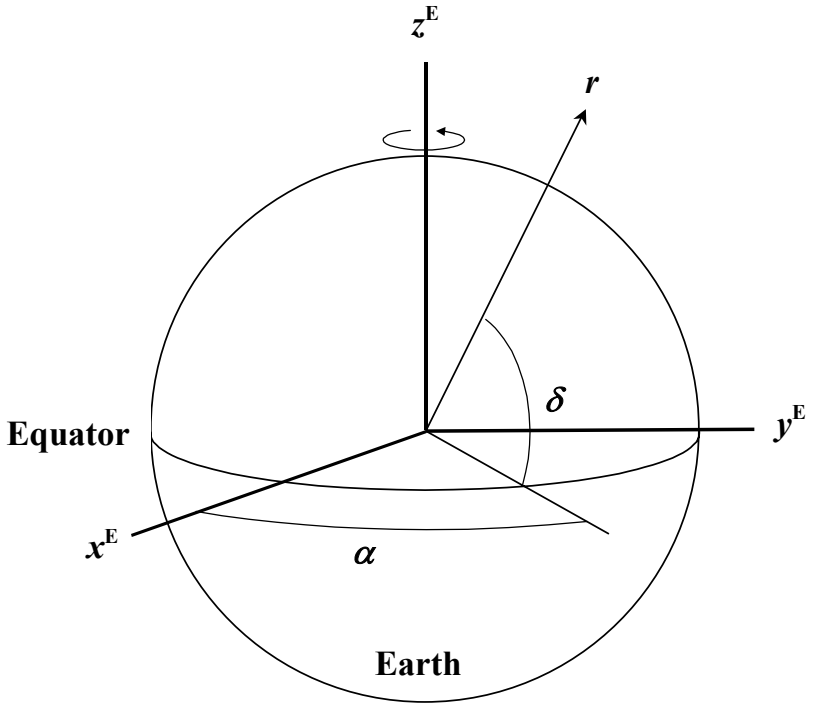

Figure 2.1 Sketch of the Earth Centered Inertial (ECI) system.
The axes are fixed in space with z along the Earth's rotation axis and x in the direction of the Vernal Equinox. The right ascension, α, and declination, δ, are also shown for an arbitrary position vector $\boldsymbol{r}$.

2.2.1 ECI & Body Frames

The first, shown in Figure 2.1, is an inertial coordinate system with origin at the center of the Earth and z-axis aligned with the Earth's rotation axis. The x and y axes lie in the Earth's equatorial plane, fixed in space so that the x-axis points toward the Vernal Equinox (loosely speaking, the Earth-Sun vector on the first day of Spring); the y-axis completes the right-handed set. This system is called the *Earth Centered Inertial* system and will be referred to as the "ECI," or "fixed" system. Recalling the earlier discussion

on conventions, ECI frame objects will be expressed with a superscript E when discernment is necessary. Also note the longitudinal and latitudinal angles for the ECI frame are specifically designated *right ascension* and *declination* respectively.

Of course, the ECI frame is not truly inertial. For one, the frame accelerates as it travels with the Earth on its trip around the Sun. More subtly, there exists a secular (albeit tiny) motion of the line of equinoxes and a wobbling spin axis. However, the former is merely an issue of decoupling angular momentum components (discussed momentarily), and the latter is on a scale that is inconsequential to our work. For the record, the ECI frame we use conforms to the *True Equator, Mean Equinox of Epoch (J2000)* convention.[4]

The second system of axes is the *Body Frame*, denoted by a superscript B. This is a right-handed system fixed to the Lageos satellite with origin at its center of mass (CM^B), which, fortunately, also happens to be at the geometric center of the satellite. The *z*-axis is along the direction of rotational symmetry, called the *body axis* or *axial* direction. The *x* and *y* axes are fixed in the equatorial plane of the satellite but otherwise arbitrary. Due to symmetry, equatorial directions are homogeneous so it will often suffice to refer to the equatorial components generically as the *transverse* direction (or axis) rather than to the individual *x* and *y* axes. A picture of this frame is shown in Figure 2.2 on page 31.

---

[4] The *z*-axis is instantaneously aligned with the true-of-date spin axis (and hence, the equator is the true equator), while the *x*-axis points to the mean Vernal Equinox for J2000.

The angular velocity vector, $\boldsymbol{\omega}$, describes the spin of the satellite. The direction is the instantaneous axis of rotation,[5] and the magnitude is the rate of spin. While easier to conceptualize as an ECI vector, it is more convenient notationally to adopt the convention of $\boldsymbol{\omega}$ as native to the body frame. When possible to do so, the body z axis is chosen so that ω_3 is positive. It remains to specify the relationship between the body and ECI frames and, from this, derive a representation for $\boldsymbol{\omega}$.

2.2.2 Euler Angles

There are a number of ways to define the relationship between the ECI and body coordinate systems—Direction Cosines, Quaternions, Roll-Pitch-Yaw, and Euler Angles to name a few (see e.g., Hughes [1986], Chobotov [1991], Wiesel [1989], Shabana [2001], and, of course, Goldstein [1980]). Each of these conventions has advantages and drawbacks and may be more or less suitable for a particular application.

For rotational dynamics, Euler Angles are particularly convenient because of the immediate correlation with the precession, nutation and spin of the satellite. The disadvantages, however, of Euler angles are twofold. First, there is an artificial singularity when the precession angle goes to zero, making the nutation and spin angles indistinguishable. Second, some of the other conventions—quaternions in particular—

[5] Rotations are viewed in a right-handed sense. When looking back down toward the body from the positive $\boldsymbol{\omega}$ direction, the rotation is seen to be counter-clockwise about the $\boldsymbol{\omega}$ axis.

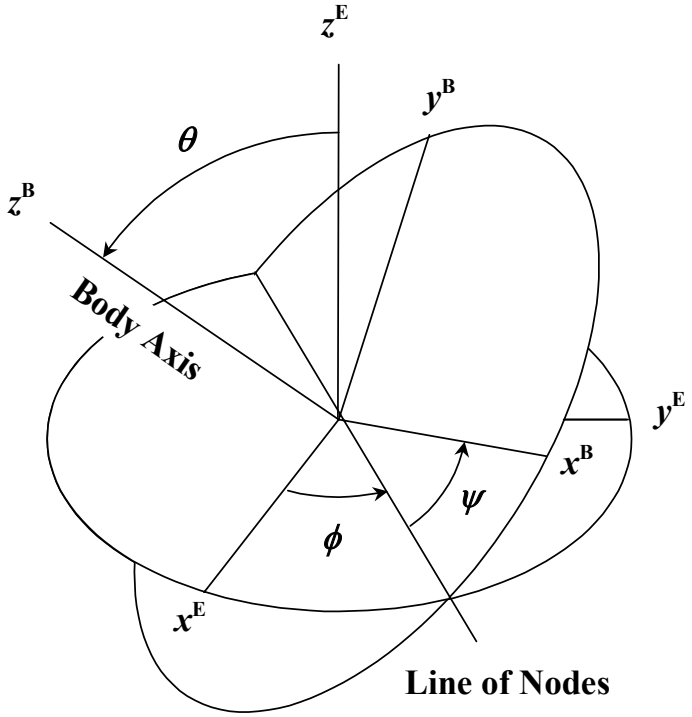

Figure 2.2 Angular relationship between the ECI and Body coordinate systems.

The rotations are done in the order ϕ, θ, ψ and these angles are called Euler Angles. The body axis is the axis of rotational symmetry for the Lageos satellite, and the line of nodes is the intersection between the respective equatorial planes of the two systems.

lead to more computationally efficient forms of the equations of motion. Nevertheless, we have chosen to proceed with the Euler angle formulation because of the intuition they provide, the immediate correlation with orbit parameters (also an Euler type rotation), and the inherited legacy of the model. Additionally, the motion of the spin axis is such that, in the short term, the nutation angle remains bounded away from the singularity. This will change as the satellite continues to lose energy so the issue will have to be revisited in the future.

The Euler angles describe the transformation via a sequence of rotations between a fixed set of axes (ECI in this case) and a rotating set of axes (body frame). The sequence, shown in Figure 2.2, is

1. A rotation of ϕ about the ECI z axis.
2. A rotation of θ about the *line of nodes*.
3. A rotation of ψ about the body z axis.

Each of these is a simple Givens Rotation [Meyer, 2000] and the product, formed left to right, has a matrix representation

$$\boldsymbol{T}^{\mathrm{EB}} = \begin{pmatrix} \cos\psi\cos\varphi - \cos\theta\sin\varphi\sin\psi & \cos\psi\sin\varphi + \cos\theta\cos\varphi\sin\psi & \sin\psi\sin\theta \\ -\sin\psi\cos\varphi - \cos\theta\sin\varphi\cos\psi & -\sin\psi\sin\varphi + \cos\theta\cos\varphi\cos\psi & \cos\psi\sin\theta \\ \sin\theta\sin\varphi & -\sin\theta\cos\varphi & \cos\theta \end{pmatrix} \quad 2$$

where the superscript identifies this as a transformation from the ECI to the body frame, i.e., $(\boldsymbol{r}^{\mathrm{E}})^{\mathrm{B}} = \boldsymbol{T}^{\mathrm{EB}} \cdot \boldsymbol{r}^{\mathrm{E}}$. This is an orthogonal transformation so the inverse transformation is given simply by the transpose

$$\boldsymbol{T}^{\mathrm{BE}} = \left(\boldsymbol{T}^{\mathrm{EB}}\right)' \quad 3$$

An observation that will prove useful later is that the columns of $\boldsymbol{T}^{\mathrm{EB}}$ are the body frame representations of the ECI basis vectors, $(\boldsymbol{e}_i^{\mathrm{E}})^{\mathrm{B}}$, while the rows are the ECI representations of the body frame basis.

The satellite's angular velocity can be determined from the Euler angle rates using trigonometry. In the body frame, $\dot{\psi}$ has only an axial component while $\dot{\theta}$ is a transverse motion; $\dot{\varphi}$ has components in both directions and so we see that

$$\boldsymbol{\omega} = \begin{pmatrix} \dot{\theta}\cos\psi + \dot{\varphi}\sin\theta\sin\psi \\ -\dot{\theta}\sin\psi + \dot{\varphi}\sin\theta\cos\psi \\ \dot{\varphi}\cos\theta + \dot{\psi} \end{pmatrix} \tag{4}$$

2.2.3 Inertia and Angular Momentum

Angular momentum is the rotational analog to linear momentum, $\boldsymbol{p}$, and is given by

$$\boldsymbol{L} = \boldsymbol{r} \times \boldsymbol{p} \tag{5}$$

Strictly speaking, a bit of caution is required here. While (5) is convenient notation, it masks the interesting dynamics. Better is the differential form where each element of mass has a corresponding position and velocity so that

$$d\boldsymbol{L} = \boldsymbol{r}_{dm} \times d\boldsymbol{p} \tag{6}$$

This can then be integrated over the body to give the total angular momentum of the system.

We have assumed from the beginning that only the spin angular momentum need be considered in (1) and now, (6). This is justified with the choice of the satellite center of mass as the origin of the local frame. It can easily be shown that (6) separates nicely into a problem of the translational motion of the center of mass (i.e., the orbital motion) and the rotation of the satellite about the mass center. Generally, the system potentials can be similarly segregated. Therefore, the problems of translational and rotational motion can be treated independently (Goldstein [1980]). As an aside, this same line of reasoning governs the separation of the Earth system's orbital motion about the Sun and the local rotational motion (such as orbiting satellites) thereby justifying the "inertial" in ECI.

Having established this initial viewpoint, a partial refinement is necessary. Hughes [1986] points out that a number of high-order environmental forces have cross-over effects between the spin and orbit motions, so called *orbit-attitude coupling*. Indeed, this point was well established in Chapter 1. Nevertheless, it is generally sufficient to separate the problem as above and treat coupled effects as perturbations after the fact.

Proceeding with 6, Chobotov [1991] has shown

$$\boldsymbol{L} = \int_{\mathrm{B}} \boldsymbol{r} \times (\boldsymbol{\omega} \times \boldsymbol{r}) dm = \left(\int_{\mathrm{B}} (r^2 \boldsymbol{1} - \boldsymbol{rr}) \, dm \right) \cdot \boldsymbol{\omega} \qquad 7$$

where the integral is over the spacecraft body, and *dm* is a differential mass element. The remaining integral term on the right side of (7) is the *inertia tensor*,

$$\boldsymbol{I} = \int_{\mathrm{B}} (r^2 \boldsymbol{1} - \boldsymbol{rr}) \, dm \qquad 8$$

which depends both on the mass properties of the body and on the choice of body axes. It is the rotational analog to mass. Thus,

$$\boldsymbol{L} = \boldsymbol{I} \cdot \boldsymbol{\omega} \, . \qquad 9$$

The matrix representation of $\boldsymbol{I}$ is hermitean and so diagonalizable

$$\widetilde{\boldsymbol{I}} = I_1 \boldsymbol{e}_1 \boldsymbol{e}_1 + I_1 \boldsymbol{e}_2 \boldsymbol{e}_2 + I_3 \boldsymbol{e}_3 \boldsymbol{e}_3 \, . \qquad 10$$

In turn, this implies there is a set (or sets) of axes for which $\boldsymbol{I}$ is already diagonal. These are called *principal axes* and can conceptually be regarded as the directions of mass-symmetry within the body. The body axes of Lageos are such a set.

2.2.4 Equations of Motion

Turning our attention back to (1), we may now write

$$\boldsymbol{I}\cdot\dot{\boldsymbol{\omega}}+\boldsymbol{\omega}\times(\boldsymbol{I}\cdot\boldsymbol{\omega})=\boldsymbol{N}\,. \tag{11}$$

It is immediately evident that the equation is quite complicated if the body axes are not principal. This is important for Lageos in that a small imbalance within the satellite would cause a misalignment between the geometric symmetries and the (true) principal axes. To our knowledge, no one has addressed this issue which, if true, could represent an important and significant error source in the modeling effort. This concern is revisited in Chapter 4.

For now, however, we proceed under the previous assumption, and (11) simplifies nicely. In fact, Lageos is *axisymmetric* (see Section 2.3.1) so the two transverse moments are equal, $I_1 = I_2$. With this in mind, the equations in scalar form become

$$\begin{aligned} I_1\dot{\omega}_1 &= -\omega_2\omega_3(I_3-I_1)+N_1 \\ I_1\dot{\omega}_2 &= \;\;\omega_1\omega_3(I_3-I_1)+N_2\,. \\ I_1\dot{\omega}_3 &= \;\;N_3 \end{aligned} \tag{12}$$

To obtain an expression in terms of the Euler angles, we substitute (4) and the derivative of (4) into (12), then simplify. The result is a somewhat complicated second order differential equation in the Euler angles.

The Lageos Spin State Equations of Motion

Separating the terms that involve torques for notational convenience, we identify *free response* (*fr*) and *forced response* (*F*) components to these equations of motion. And so we have, finally,

$$\begin{aligned} \ddot{\theta} &= \ddot{\theta}_{fr} + \ddot{\theta}_{F} \\ \ddot{\varphi} &= \ddot{\varphi}_{fr} + \ddot{\varphi}_{F} \\ \ddot{\psi} &= \ddot{\psi}_{fr} + \ddot{\psi}_{F} \end{aligned} \tag{13}$$

where the free response equations are

$$\begin{aligned} \ddot{\theta}_{fr} &= -\frac{\dot{\varphi}\sin\theta}{I_1}\left[(I_3 - I_1)\dot{\varphi}\cos\theta + I_3\dot{\psi}\right] \\ \ddot{\varphi}_{fr} &= \frac{\dot{\theta}}{I_1\sin\theta}\left[(I_3 - 2I_1)\dot{\varphi}\cos\theta + I_3\dot{\psi}\right] \\ \ddot{\psi}_{fr} &= -\frac{\dot{\theta}}{I_1\sin\theta}\left[\left((I_3 - I_1)\dot{\varphi}\cos\theta + I_3\dot{\psi}\right)\cos\theta - I_1\dot{\varphi}\right] \end{aligned} \tag{14}$$

and the force response equations are

$$\begin{aligned} \ddot{\theta}_{F} &= \frac{1}{I_1}\left[N_1\cos\psi - N_2\sin\psi\right] \\ \ddot{\varphi}_{F} &= \frac{1}{I_1\sin\theta}\left[N_1\sin\psi + N_2\cos\psi\right] \\ \ddot{\psi}_{F} &= -\frac{\cos\theta}{I_1\sin\theta}\left[N_1\sin\psi + N_2\cos\psi\right] + \frac{N_3}{I_1} \end{aligned} \tag{15}$$

Equations 13, 14, and 15 completely describe the rotational motion of Lageos. With proper evaluation of the torques, very precise results should be obtainable.

Unfortunately, there are many hurdles to achieving the kind of accuracy desired when computing torques, but that comes later.

Implementation

The Euler equations of motion described by (13) through (15) are a system of second order, autonomous non-linear ordinary differential equations (ODEs). We wish to propagate the system in time given an initial state for the Euler angles and their rates. At first glance, this system appears quite manageable—there are only a handful of variables, and the statement of the problem is relatively clean. However, non-linearity and the presence of forcing terms (torques) prohibit an analytical treatment of the problem. Evaluation of the torques represents a set of problems within the problem, and it is the central goal of our effort to quantify their effects. Nevertheless, once $\boldsymbol{N}$ is determined, the equations of motion still must be resolved.

The issues of numerical implementation are addressed later; for now, we simply exhibit the reduction of the equations of motion to a form for use within the model. The equations of motion are restated as a first order system by setting

$$\boldsymbol{Y} = (\theta \quad \phi \quad \psi \quad \dot{\theta} \quad \dot{\phi} \quad \dot{\psi}) \tag{16}$$

so that

$$\dot{\boldsymbol{Y}} = \begin{pmatrix} \dot{Y}_1 \\ \dot{Y}_2 \\ \dot{Y}_3 \\ \dot{Y}_4 \\ \dot{Y}_5 \\ \dot{Y}_6 \end{pmatrix} = \begin{pmatrix} Y_4 \\ Y_5 \\ Y_6 \\ \ddot{\theta}_{fr} + \ddot{\theta}_F \\ \ddot{\varphi}_{fr} + \ddot{\varphi}_F \\ \ddot{\psi}_{fr} + \ddot{\psi}_F \end{pmatrix} \tag{17}$$

where

$$\begin{aligned} \ddot{\theta}_{fr} &= -\frac{Y_5 \sin Y_1}{I_1}\left[(I_3 - I_1)Y_5 \cos Y_1 + I_3 Y_6\right] \\ \ddot{\varphi}_{fr} &= \frac{Y_4}{I_1 \sin Y_1}\left[(I_3 - 2I_1)Y_5 \cos Y_1 + I_3 Y_6\right] \\ \ddot{\psi}_{fr} &= -\frac{Y_4}{I_1 \sin Y_1}\left[\left((I_3 - I_1)Y_5 \cos Y_1 + I_3 Y_6\right)\cos Y_1 - I_1 Y_5\right] \end{aligned} \tag{18}$$

and

$$\begin{aligned} \ddot{\theta}_F &= \frac{1}{I_1}\left[N_1 \cos Y_3 - N_2 \sin Y_3\right] \\ \ddot{\varphi}_F &= \frac{1}{I_1 \sin Y_1}\left[N_1 \sin Y_3 + N_2 \cos Y_3\right] \\ \ddot{\psi}_F &= -\frac{\cos Y_1}{I_1 \sin Y_1}\left[N_1 \sin Y_3 + N_2 \cos Y_3\right] + \frac{N_3}{I_1} \end{aligned} \quad . \tag{19}$$

These may be computed efficiently as follows

1. Compute: $sY_1 = \sin Y_1$, $cY_1 = \cos Y_1$, $I = I_3/I_1$, $I' = 1 - I$
2. Set $w_1 = Y_5\, cY_1$, $w_2 = I\, Y_6$, $w_3 = I'\, w_1 - w_2$, $w_4 = Y_4/sY_1$
3. Determine N_1, N_2, and N_3
4. Compute $sY_3 = \sin Y_3$, $cY_3 = \cos Y_3$

5. Set $w_5 = \frac{1}{I_1 \, sY_1}(N_1 \, sY_3 + N_2 \, cY_3)$

6. Then

$$\dot{\boldsymbol{Y}} = \begin{pmatrix} Y_4 \\ Y_5 \\ Y_6 \\ Y_5 \, sY \, w_3 + \frac{1}{I_1}(N_1 \, cY_3 - N_2 \, sY_3) \\ -w_4(w_3 + w_1) + w_5 \\ w_4(w_3 \, cY_1 + Y) - cY_1 \, w_5 + \frac{N_3}{I_1} \end{pmatrix} \qquad 20$$

2.3 The Lageos Satellite

While some information about the Lageos satellite has already been discussed, the following is a comprehensive summary. There are various sources for much of the data presented. They are summarized here in lieu of littering the following text with redundant citations:[6] Habib et al [1994], Kheyfets [1992, 1993], Avizonis [1997], Rubincam [1987], Bertotti & Iess [1991], as well as [A], [B], [C], [D], [E], and [H].

2.3.1 Spacecraft Properties

An extremely accurate orbit is required for Lageos to function as a space bound reference for high precision measurement of geodetic phenomena. The design of the satellite

[6] Regrettably, we were unable to acquire the apparent common source document for Lageos structural information, Johnson et al [1976]; although, we list it in the references to be thorough.

balances the objectives of maximizing the body's reflective properties while minimizing its susceptibility to orbit perturbing forces.

To accomplish this, Lageos was given a spherical shape and a high mass to area ratio. The spherical geometry provides attitude independence and a large surface area compared to the cross-sectional footprint. An extra-atmospheric orbit altitude was chosen so that drag is negligible. *Axisymmetry*, i.e., the body has only two independent principal inertial directions, provides a stable spin axis to store energy. Finally, materials were chosen to minimize the influence of the Earth's magnetic field.

Lageos consists of a spherical aluminum shell wrapped around a cylindrical beryllium-copper core. The shell is constructed of two 30 cm radius hemispheres of 6061 aluminum bolted together by a tension stud. The external surface is covered with 426 cube corner reflectors (422 fused silica glass, 4 germanium) each 3.8 cm in diameter, giving it the appearance of a giant golf ball. The literature is unclear as to the thickness of the shell; although, there are some indications that the cavity is just large enough to house the brass core. This is important in that, while generation of magnetic field induced currents seems improbable at the surface due to the presence of the reflectors, one can surmise a boundary layer immediately beneath the reflectors inside of which such currents may be possible.

The beryllium-copper core measures 31.76 cm in diameter and 26.70 cm tall. It is connected symmetrically to the tension stud, i.e., the body axis of the satellite. Disagreement exists in the literature on the core's material make-up—a number of

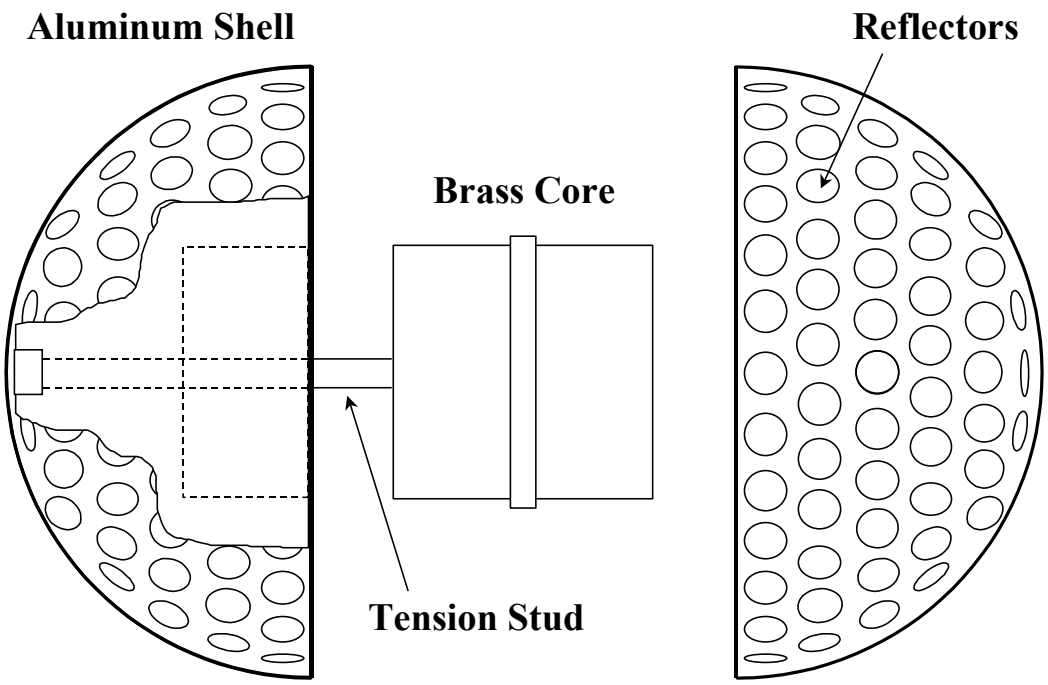

Figure 2.3 Notional schematic of the Lageos satellite structure

sources refer to it as brass. However, Rubincam [1987], citing Johnson et al [1976] as the authority, directly refutes this and categorically claims the beryllium-copper makeup.

The cylindrical core gives Lageos a concentration of mass along the body axis while maintaining rotational symmetry. The result is axisymmetry with principal axes along and orthogonal to the body axis. The corresponding moments of inertia are $I_3 = 1.314\times10^8$ g cm^2 and $I_1 = 1.271\times10^8$ g cm^2. The benefit of this construction is the ability to "spin-up" the satellite at the beginning of life and store energy as spin angular momentum to provide additional resistance to perturbations. Table 2.1 summarizes the properties of the Lageos spacecraft.

Table 2.1 Key structural properties of the Lageos Satellite

Exterior Shell	
Geometric Shape	Spherical – two hemispherical shells
Material Composition	6061 Aluminum Alloy
Gross Dimension	60 cm diameter
Surface Features	426 cube corner reflectors 3.8 cm in diameter
Internal Structure	
General Features	Cylindrical core connected along its axis to the shell by a tension stud
Core Material Composition	Beryllium-Copper Alloy
Core Dimension	31.76 cm diameter x 26.70 cm height
Mass Properties	
Total Mass	4.11×10^{5} g
Principal Moment – Axial	$I_3 = 1.314\times10^{8}$ g cm^2
Principal Moment – Transverse	$I_1 = 1.271\times10^{8}$ g cm^2

2.3.2 The Lageos Orbit

As a passive satellite, Lageos has no mechanism for controlling attitude or correcting for orbit disturbances. Therefore, in addition to designing the spacecraft to inhibit orbit perturbations, the orbit itself was chosen to maximize orbit tracking accuracy. These decisions have helped make Lageos the most accurately tracked satellite, yielding three-day root-mean square fits of better than 2 cm [D]. To better describe Lageos' orbit properties, and to establish some mathematical conventions used in the model, a brief detour is necessary to discuss the source of our orbital data for Lageos and review some basic orbit related terminology.

The orbit data for Lageos is made available courtesy of Kelso [1998] who maintains a complete repository of ephemerides for most of the non-classified satellites tracked by NORAD (North American Aerospace Defense Command) is available at [F]. The data, called the *NORAD Two Line Element Sets* (2LES), uses the standard Keplerian set of elements to depict the orbit,[7] which can be described as elliptical motion in a given plane (the *orbit plane*). The Keplerian elements locate the orbit plane in inertial space—*inclination* "i", *right ascension of the ascending node* or *raan* "Ω"; describe the ellipse in the orbit plane—*argument of perigee* "ω", *semi-major axis "a*", *eccentricity* "e"; and specify the satellite's motion—*mean anomaly* "M", *mean motion* "n" (see, e.g., Roy [1988] or Danby [1992]). Kepler's third law relates the mean motion to the semi-major axis so the two are redundant. Therefore, only n is explicitly provided in the 2LES.[8]

---

[7] It is often mistakenly assumed that the Keplerian elements describing the orbit of a particular satellite are universal. In fact, this is not the case. Element sets for orbiting satellites are the result of fitting predictions from a specific model—the SGP4/SDP4 model in the case of 2LES data—to the observed data (i.e., a nonlinear optimization is done to find the parameters that generate the best fit to the data). The elements derived from a given observation are usually weight-averaged with previous solutions to reduce the variability from one set to the next. For NORAD 2LES, updated element sets are only released when they differ from the previous set by more than a threshold amount. It should be understood then, that the data we report here is taken from the 2LES and so is specific to the SGP4/SDP4 model. Generally speaking, these values cannot be plugged directly into a different orbit model. For more on this, see Kelso [1998].

[8] The 2LES data is formatted so that it may also be used with the simpler SGP model in addition to the full SGP4/SDP4 version. The latter "recovers" its self-consistent mean motion and corresponding semi-major axis internally (Hoots and Roehrich [1980]). We have analyzed both versions and determined, perhaps not too surprisingly, that the "raw" mean motion explicit in the 2LES is a better choice for simplified work.

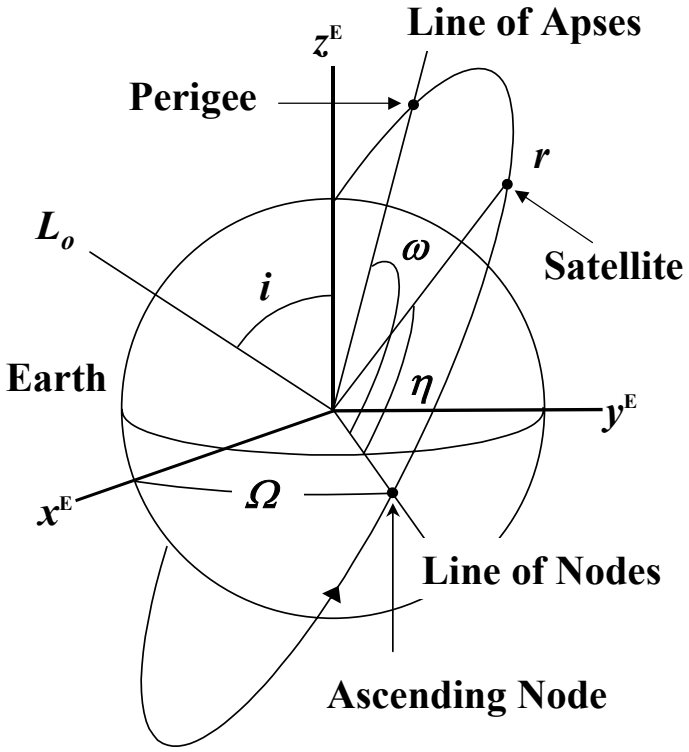

Figure 2.4 Graphical definition of the Keplerian orbit parameters

Visual depiction of Keplerian orbit parameters and related definitions. The parameters are referenced to the Earth Centered Inertial (ECI) system. Also shown is the orbital angular momentum, $\boldsymbol{L}_o$, which is normal to the orbit plane. The remaining features are explained in the text.

Figure 2.4 shows the spatial relationship between the orbital system and the ECI frame. It can be seen that the three angles, Ω, i, and ω are, in fact, Euler angles playing the respective roles of ϕ, θ, and ψ. Also shown is the variable η, the *net angular position*, i.e., the angular position of the satellite in the orbit plane relative to the line of nodes. The *ascending node* is the point of the satellite's south-to-north crossing of the equator. The line through this point and the origin is called the *line of nodes*. For

elliptical motion, *perigee*[9] is the point in the orbit of closest approach to the mass center and lies on the major axis, also called the *line of apses*. Completing the review of orbit related terminology, note that M is a pseudo-position angle that changes uniformly with time by $M = n(t–t_o)$ and so differs periodically from the true angular position (*true anomaly*).

It is typical to think of Keplerian parameters as constants (excepting M which is a type of instantaneous "position") but, in fact, the orbit parameters are dynamic (e.g., the aforementioned secular decay in Lageos' semi-major axis). It can therefore be misleading to present a table of constant Keplerian elements (or their linear rates of change) as representative of the orbit for all time, or even for a period of time. However, the secular change of these parameters for Lageos is quite small, and so we feel comfortable providing the nominal orbital elements in Table 2.2 (page 46), which were derived by applying constant or linear best-fit approximations to post 1990 2LES data (see Section 3.2).

2.4 Summary

Embedded in the equations of motion (1) are numerous complexities requiring resolution before progress can be made toward a solution. In particular, the general problem must be formulated in terms specific to the Lageos spin dynamics system. To that end, we

[9] This is an Earth-specific term, the more general term is *pericenter*.

have established essential conventions and a mathematical framework, described the pertinent dynamical issues, and recast the equations of motion in a form suitable to the Lageos spin state modeling effort. The important physical and orbital characteristics of the Lageos satellite have also been identified to facilitate the development of the specific model components. With this foundation in place, we proceed with the detailed construction of the Lageos spin dynamics model.

Table 2.2 Nominal Lageos orbit parameters

Values derived from the NORAD Two-Line Element Sets data for the period since 1990. Semi-major axis, eccentricity, and inclination are mean values for the period. The angular measures—argument of perigee, right ascension of the ascending node, and mean anomaly—are derived from linear best-fit approximations.

Orbit Parameter	**Symbol**	**Nominal Value**
Semi-Major Axis	a	1.22712×10^{9} cm
Eccentricity	e	0.00443
Inclination	i	109.84°
Argument of Perigee	ω	
J2000 Value	ω_o	211.82°
Precession period of the Line of Apses	T_ω	1681.6 JD
Right Ascension of the Ascending Node	Ω	
J2000 Value	Ω_o	109.05°
Precession period of the Line of Nodes (“orbital precession”)	T_Ω	1050.9 JD
Mean Anomaly	M	
J2000 Value	M_o	107.68°
Mean Motion	n	6.38665 rev/JD

3 The Lageos Spin Model

3.1 Overview

We have previously summarized the evolutionary track of the Lageos Spin Dynamics model first introduced by Habib et al [1994] and later modified slightly by us [Williams, 1997]. In the ensuing remarks, we shall refer often to these models, as well as their revision as presented here. For clarity in the discussion, we designate the previous (combined) efforts as the *H&W model* and the current effort as the *W02* model. It usually will not be necessary to distinguish between the original efforts of Habib et al in 1994 and our own in 1997, but we will clarify when appropriate.

Recall that while the H&W model yields decent results and qualitatively captures the spin state behavior, it still falls far short of providing reliable and specific predictive results. From the standpoint of dynamical modeling, therefore, our primary goal for this effort has been to improve the predictive nature of the existing model.

The thrust of our work also necessitated modifications to the software package. Considerable effort was expended improving features that may only impact the quality of the dynamical results indirectly, if at all. Enhancements include the type of data sets that

are generated, analysis and monitoring of numerical integration routines, optimizing code for efficiency, and incorporating a nonlinear optimization package for model parameters.

Most of our work derived not from any specific intent, but from a general bottom-up approach to the modeling effort. In particular, our goal was to critically examine every piece of the H&W model (physical, software, etc) seeking opportunities for improvement. Of course, the effort was tempered by the need to refrain from computational excess (the model must be run in finite time!).

For the model's physical aspects, we introduced modifications that more closely approximate Lageos' true environment, regardless of the pre-conceived notion of the impact on the predicted dynamics. In some cases, these efforts were rewarded; in others, not much response was observed. Either way, the work succeeds by providing empirical evidence of the impact (or lack thereof) due to a particular enhancement. The remainder of this chapter details each of the issues that captivated our attention in the revision process.

3.1.1 Errors and Basic Modeling Issues

Modeling error is referred to throughout the discussion. This is a generic reference to the difference between our spin state prediction results and the idealized 'true' spin state of the satellite. Avizonis' data is used as a 'truth' proxy for comparison purposes, but it represents a limited data set and is subject to errors of its own. Therefore, the 'truth' target we seek is often more abstract than concrete.

A useful perspective is to view model output as the sum of the system's true response and *artificial effects* introduced by the model. Artificial effects arise from both the model's imperfect approximation of the physical system (*model fidelity*) and the inherent limitations of numerical simulation (*model implementation*).

Some of the model fidelity related issues are:

- Absence of 'real-world' effects we neglect;
- Perturbations of modeled effects introduced by making simplifying assumptions;
- Errors and uncertainties in model parameters and initial conditions.

Issues related to model implementation include:

- Poor conditioning (an inherent property of the system);
- Floating point arithmetic error (intrinsic deficiency of computer implementation);
- Numerical integration error (property of the integrator; occurs even for perfect arithmetic and conditioning).

Some of these 'artificial effects' persist no matter how detailed the modeling effort, and so the results will always have some error. In fact, there is sometimes a direct trade-off between fidelity and implementation concerns, as the latter tend to be sensitive to the complexity of the system. Consequently, we focused on both improving the fidelity of the model and ensuring the numerical concerns, particularly the integration package, are addressed.

3.2 Orbit Propagation Model

Lageos' response to the space environment is dependent on its location in that environment. Therefore, an orbit propagation module is needed in the model. Presumably, the impact of "small" position determination errors on the spin state propagation will be negligible. Of course, "small" is ill defined—how small? Moreover, the response may not be negligible if the "small" position errors are biased rather than random. Still another issue is the degree to which present and future improvements elsewhere in the model are impacted by the fidelity of the satellite position determination. With this in mind, we examine the implementation of the orbit module in the H&W model and explore whether possible improvements generate enough impact to justify additional computational complexity.

Use of a full dynamical orbit propagation model, such as the SGP4/SDP4 model used by NORAD for the 2LES data, is not a serious consideration. For one thing, while propagation from a set of elements achieves locally precise results, significant errors can accumulate over longer extrapolations. It would therefore be necessary to incorporate the full 2LES database to ensure only local propagation is performed (element set to element set). More importantly, the numerical cost of this approach (with or without the database) is disproportionately expensive compared to the accuracy gained over simpler ideas.

3.2.1 Simple Orbit Model

Based on the results in Chapter 2, Lageos' orbit can be well approximated by a purely circular motion. This is the approach taken in the H&W model. The orbit is spatially located by a, i, $\Omega(t)$, and $\eta(t)$. The semi-major axis, a, is now simply the orbital radius; the time dependence (linear by assumption) of the right ascension of the ascending node and the net angular position is explicitly indicated. We define $\dot{\Omega}$ as the orbital precession rate and use mean motion to determine angular position (motion is uniform for a circular orbit). The simplified orbit model is thus given by

$$\begin{aligned} a \\ i \\ \Omega &= \dot{\Omega}t + \Omega_o \\ \eta &= nt + \eta_o \end{aligned} \qquad 21$$

requiring six parameters to be specified.

To determine the orbit model parameters, we revisit the 2LES data. The plots in Figures 3.1 and 3.2 show historical values since 1990 of the relevant Keplerian elements. While variation is present in the data, the effects are quite small. For example, the periodic oscillation in inclination has a half-amplitude of about 1¾′; at Lageos' altitude that amounts to ~6 km (at apex) spatial error. Likewise, even with secular decay of the semi-major axis evident, the mean decrease over the data set is only tens of meters.

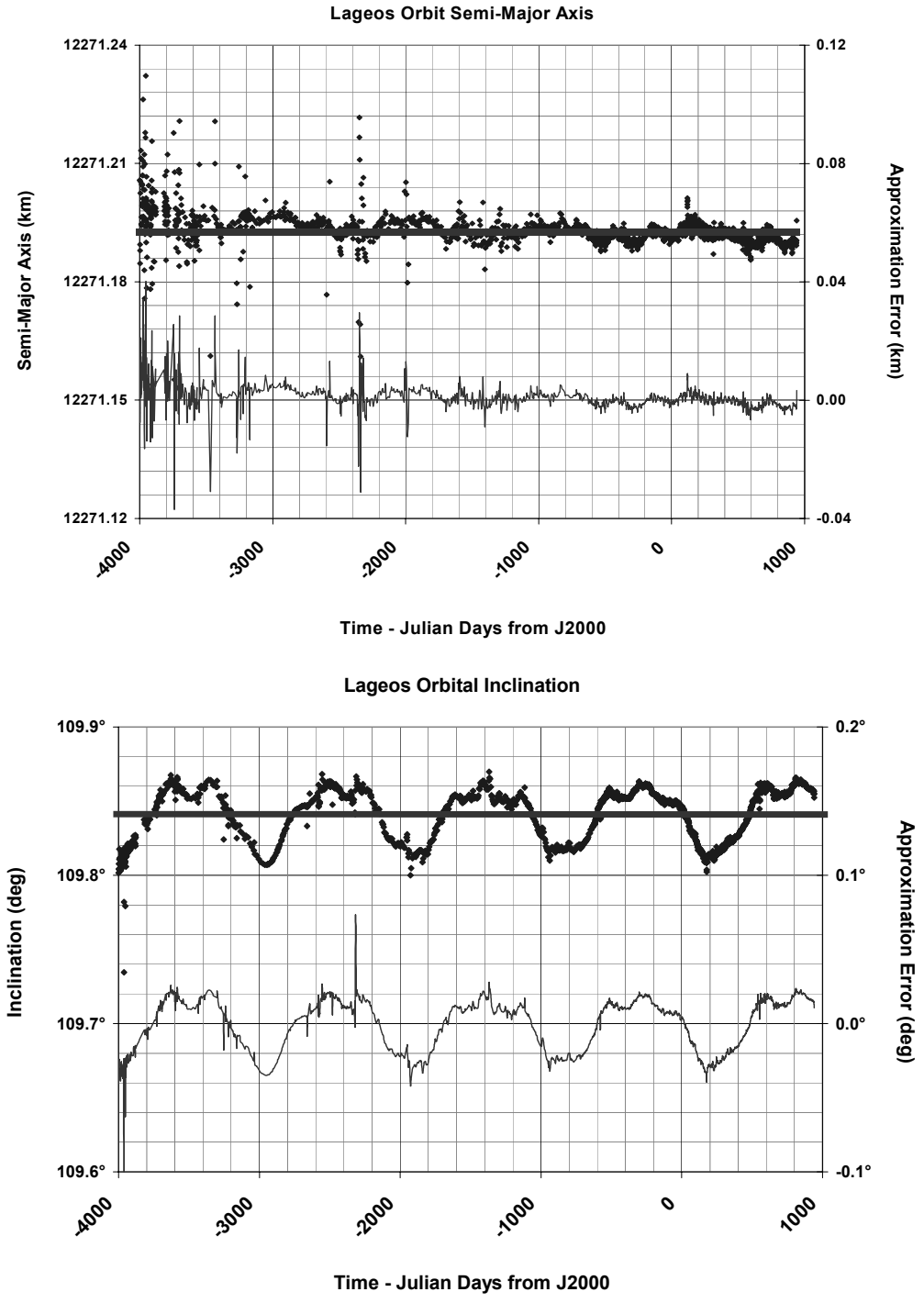

Figure 3.1 Historical values of Lageos orbit semi-major axis and inclination.

Data values are shown as the scatter of diamonds in the upper half of the graphs. The simple approximations used for the model are shown as overlaying solid lines, and the corresponding error for the approximations are shown in the bottom portion of the plots.

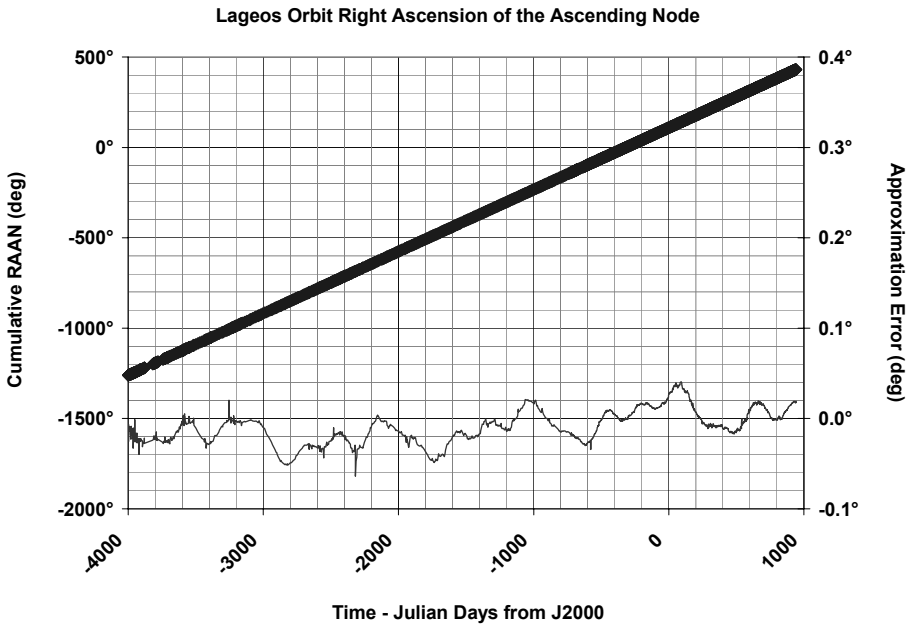

Figure 3.2 Historical values of Lageos orbit right ascension of the ascending node

The data shows a clear linear trend. The error corresponding to the best-fit linear approximation is shown in the bottom portion of the plot.

The situation for the rate values ($\dot{\Omega}$ and n) is perhaps slightly more complicated because errors magnify with time. The scales, however, remain small. For $\dot{\Omega}$ (slope of the linear approximation to Ω), the resulting error in Ω is only modestly larger than that for inclination. Similarly, the model value for n departs from the true values by $O(10^{-5})$ revs/day. Even for a persistent bias, this totals less than 1.5°/yr.

The values M_o and η_o at $t = 0$ are dependent on the time convention used. Local time (in seconds) was the format in the H&W model with $t = 0$ corresponding to the set of initial conditions. The W02 model was updated to utilize the JD2K time format. This releases us from re-evaluating time-based initial values for integrations starting from

different epochs. Table 3.1 summarizes the orbit model parameter values corresponding to (21).

Table 3.1 Parameter values for the simple orbit model

a	1.22712×10^9 cm
i	109.84°
Ω_o	109.05°
$\dot{\Omega}$	0.3425558 °/JD
η_o	107.68°
n	6.386646 rev/JD

Careful observers may have detected an omission. While circular motion can certainly be described using mean motion, the net angular position of Lageos based on the 2LES data is $\eta = M + \omega$,[1] not the $\eta = M$ implicit in (21). This was an oversight of the H&W model, and the resulting orbit-track errors are not easy to ignore. Accounting for the true net angular position, the adjusted mean motion becomes $n' =$ 6.386052 revs/day which differs from the original by ~0.2 °/day (about 78 °/yr). To the extent that effects on the spin axis 'smooth out' over multiple orbits, this error may not have significant impact. Nevertheless, it is an easily correctable mistake that may be meaningful as higher precision results are sought.

Circularizing the orbit also places the orbit-center at a focus of the ellipse describing Lageos' true orbit. The result is a mean bias along the positive semi-major axis of about 82 km. While interesting, we do not believe this otherwise plays a meaningful role.

[1] Technically ω is not defined for circular motion so this equation is an abuse of notation. However, n only accounts for the motion of M which itself refers to the line of apses. In order to have η represent the net angular position from the line of nodes, ω must be accounted for in the model's mean motion.

3.2.2 Orbit Model Enhancements

After correcting for the error in the mean motion, the results obtained above for a circular orbit seem reasonable enough. However, we have introduced a few further enhancements at only a modest additional cost. In particular, we assume quadratic rather than linear time dependence for η and relax the assumption of a circular orbit.

The 2LES data is derived from observations always taken at the ascending node of the orbit. For this data the modulated value of

$$\eta = M + \omega \tag{22}$$

should be very close to zero, deviating only to the extent M differs from the *true anomaly* (true instantaneous angular position with respect to the line of apses). These results are verified in Figure 3.3. Since M differs from true anomaly by at most half a degree, we continue to allow it to approximate true anomaly and so keep η in the form of (22). Using the 2LES revolution numbers, we are able to determine the monotonic data set for (22) and so apply a quadratic approximation for use in the model.[2]

It remains to determine the instantaneous radial distance for the elliptical orbit. This is given by

$$r = a(1 - e\cos E) \tag{23}$$

---

[2] It is elementary to also generate a sinusoidal correction to account for the mismatch between the mean and true anomalies. We provide such an option in the model but don't recommend it due to the complexity/cost vs. benefit gained.

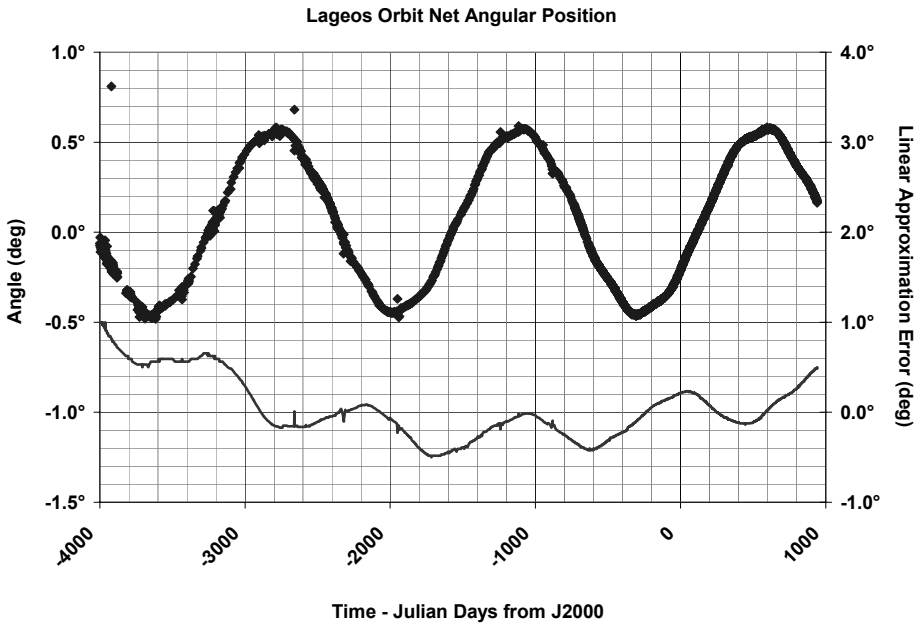

Figure 3.3 Historical values of Lageos orbit net angular position

Lageos' net angular position, η, is approximated by $M+\omega$. The historical modulated values of $M+\omega$ are shown in the top portion of the graph. Data sets are always recorded as Lageos crosses its ascending node ($\eta = 0$); therefore, the observed zero-mean sinusoidal variance is the error in using mean anomaly to approximate true anomaly. The error corresponding to a linear fit of the monotonic net angular position data (not shown) is plotted on the bottom portion of the graph and evidences an underlying quadratic trend.

where E is the *eccentric anomaly* which satisfies

$$E - e\sin E = M \tag{24}$$

This can only be solved iteratively for E but for small e, the approximation

$$E \approx M + e\sin M \tag{25}$$

or even just $E \approx M$ may be used. This requires an evaluation of M, which unfortunately, cannot be extracted from the preceding work. By (23), any errors in M appear in r as $\mathrm{O}\left((\Delta M)^2\right)$; so we simply use the linear approximation implied by (21).

Implementation

Together these ideas provide the revised orbit propagation algorithm. Table 3.2 gives the parameter values we used for this approach and the algorithm proceeds as follows:

1. Input: t JD2K;
2. Constants: a, i, e
3. Compute

$$\begin{aligned} \Omega &= \dot{\Omega}t + \Omega_o \\ \eta &= \eta_a t^2 + \eta_v t + \eta_o \\ M &= nt + M_o \end{aligned} \qquad 26$$

   where the coefficients on the right-hand-side (RHS) are predetermined.
4. Set $E = M$ (or use (25)) and evaluate $r = a(1 - e\cos E)$
5. Calculate the ECI unit vector $\hat{\boldsymbol{r}}$ as the first row of (2) using Ω, i, and η in place of ϕ, θ, and ψ respectively.

The effect of these changes is a reduction of the radial position error from tens of kilometers to tens of meters and the elimination of the secular error for angular position.

Table 3.2 Revised parameter values for the modified orbit model

a	1.2271192×10^{9} cm
e	0.004432°
i	109.84°
Ω_o	109.05°
$\dot{\Omega}$	0.342556 °/JD

η_o	319.42°
η_v	2298.97906 °/JD
η_a	1.7724×10^{-7} °/JD^2
M_o	107.68°
n	2299.19273 °/JD

True, the cost of this approach more than doubles that of the simpler version above, but it remains very inexpensive compared to the overall cost of the model.

3.3 Introduction to Environmental Torques

There are two factors mutually important to modeling the torques affecting the satellite spin state: 1) the representation of the spacecraft, and 2) the representation of the environment. These factors are generally intertwined, though more in some cases than in others, so we discuss them together for each torque source.

Several environmental effects potentially impact the attitude and spin of a satellite. The most prominent are (Hughes [1986]):

- A non-uniform gravitational field over a material body—*gravitational torque*;
- The collision of atmospheric particles with the spacecraft surface—*aerodynamic torque*;
- The transfer of electromagnetic energy—*radiation torque*;
- The interaction of a metallic spacecraft body with the Earth's magnetic field—*magnetic torque*;
- The ongoing impact of micrometeoroids—*meteoroidal torque*; and
- The deformation of the spacecraft body due to a non-uniform temperature gradient—*thermoelastic effects*.

Of these, two are dominant for the Lageos spin dynamics, the gravitational and magnetic torques. The remaining factors are insignificant by comparison, though thermoelastic

effects present some compelling issues, and so are not included in the spin model. We begin with a discussion of these unmodeled effects to justify their exclusion and then move on to the gravitational and magnetic torques.

3.3.1 Unmodeled Effects

Surface Effects

Apart from the thermoelastic effects, which we will discuss momentarily, the remaining sources from the preceding list—aerodynamic torque, radiation torque, and meteoroidal torque—behave according to a similar profile. Each concerns the bombardment of the satellite surface with a source of energy, and the subsequent reaction is a function of the surface geometry. The torque induced by these effects is a function of the satellite *center of pressure*, $\boldsymbol{c}_p$,[3] and is given by

$$\boldsymbol{N}_s = \boldsymbol{c}_p \times \boldsymbol{F}_s \tag{27}$$

where $\boldsymbol{F}_s$ is the net force on the surface. However, unlike the center of mass, which is a fixed reference for a rigid body, $\boldsymbol{c}_p$ is a function of both satellite geometry and the direction of incident momentum (*angle of attack*).

There are then two conditions for negligible torque in (27): i) the magnitude of the force is small and/or ii) the force acts parallel to the center of pressure. We have already argued that Lageos' orbit places it beyond the realm of significant atmospheric disruption

[3] The center of pressure is the surface effect analog to the center of mass and is a vector quantity defined with respect to the center of mass.

thereby justifying a small force assumption. Likewise, Hughes [1986] has shown the surface pressure (force per area) due to meteoroidal impact is at least four orders of magnitude smaller than that of solar radiation pressure. However, we can actually show that the stronger result of (ii) holds for Lageos due to spherical surface symmetry.

The center of pressure is given by the surface integral

$$\boldsymbol{c}_g = \frac{1}{A_p} \oiint_B H(\cos\alpha)\cos\alpha\, \boldsymbol{r}\, dA \tag{28}$$

where $\boldsymbol{r}$ is the position of the surface element dA, and α is the corresponding angle of attack; A_p is a scalar we will not need to evaluate. $H(x)$ is the *Heaviside* function

$$H(x) = \begin{cases} 1 & x \geq 0 \\ 0 & x < 0 \end{cases} \tag{29}$$

so that the integration (28) takes place over the half-sphere facing the incident direction. The integral is symmetric about the incident direction, and so, $\boldsymbol{c}_p$ can only depend on this direction. A similar symmetry argument shows the net force also must act in the incident direction (components of force normal to the incident direction cancel when integrated over the surface). But then $\boldsymbol{c}_p$ and $\boldsymbol{F}_s$ are parallel (or anti-parallel), and therefore (27) vanishes. There is no net torque on the Lageos satellite due to surface effects.

Thermoelastic Deformation

Thermoelastic effects represent a particularly intriguing twist to our problem. They are not an independent source of torque, but rather, a perturbation of the others. For example, the effect could alter the axisymmetry of the satellite thereby directly impacting

the equations of motion (Chapter 2). This is a disturbing possibility and perhaps a good candidate to explain errors in the results. Unfortunately, accounting for thermoelastic deformation represents a significant leap in complexity and other concerns were more pressing for the present work. Therefore, this aspect of error remains unexplored but we believe it merits investigation in future work.

3.4 Gravitational Torque Model

The primary force for orbiting bodies is due to the classical $-1/r^2$ gravitational field. This force is also one of the two dominant factors in the spin dynamics of Lageos, though the reason gravity affects satellite attitude is perhaps hard to believe. Typical satellite orbital distances from the mass center of the Earth range from ~6500 km near the Earth's surface to nearly seven times that distance for geosynchronous orbits. Intuitively, one would expect the small Δr across a satellite's body to be inconsequential at those distances. Yet, it is precisely because the Earth's tug on the nearer parts of the satellite's body is infinitesimally stronger than on its further parts that gravity torques are possible. If in addition, the satellite has a spherically asymmetric mass distribution, gravitational torque will occur.

The axisymmetric Lageos is therefore subject to gravitational torque. Deriving a general expression for gravity's effect the attitude of Lageos requires a consideration of the non-uniform mass distribution of both the Earth and the satellite. To achieve a completely general solution, the gravitational tug of other "nearby" massive bodies (Sun,

Moon, etc.) must be evaluated as well. This complete expression for the torque about the center of mass of a satellite B due to the gravitational influence of N bodies B_n is given by Hughes [1986]:

$$\boldsymbol{N}_g = -G\sum_{n=1}^{N}\int_{B_n}\int_{B}\frac{\boldsymbol{s}\times\boldsymbol{R}_n}{R_n^3}dm_n dm \qquad 30$$

where $\boldsymbol{s}$ is the body frame position of the satellite mass element dm, and $\boldsymbol{R}_n$ is the relative position of dm with respect to the body n mass element dm_n. Needless to say, this is a somewhat daunting expression.

3.4.1 Primary Torque Component

Fortunately, there is a single dominant component in (30). To first order, the general problem reduces to that of a single primary (Earth), which is also taken to have a spherical mass distribution. The Earth can then be regarded as having its mass concentrated at its center (i.e. a point mass at the ECI origin). Thus, (30) becomes

$$\boldsymbol{N}_g = -\mu_e\int_{B}\frac{\boldsymbol{s}\times\boldsymbol{R}}{R^3}dm \qquad 31$$

where $\mu_e = GM$ is the *Geocentric Gravitational Constant* and $\boldsymbol{R}$ is the ECI position of dm (Figure 3.4). If $\boldsymbol{r}$ is the ECI position of CM^B (i.e., body frame origin), then $\boldsymbol{R} = \boldsymbol{r} + \boldsymbol{s}$ and we may write

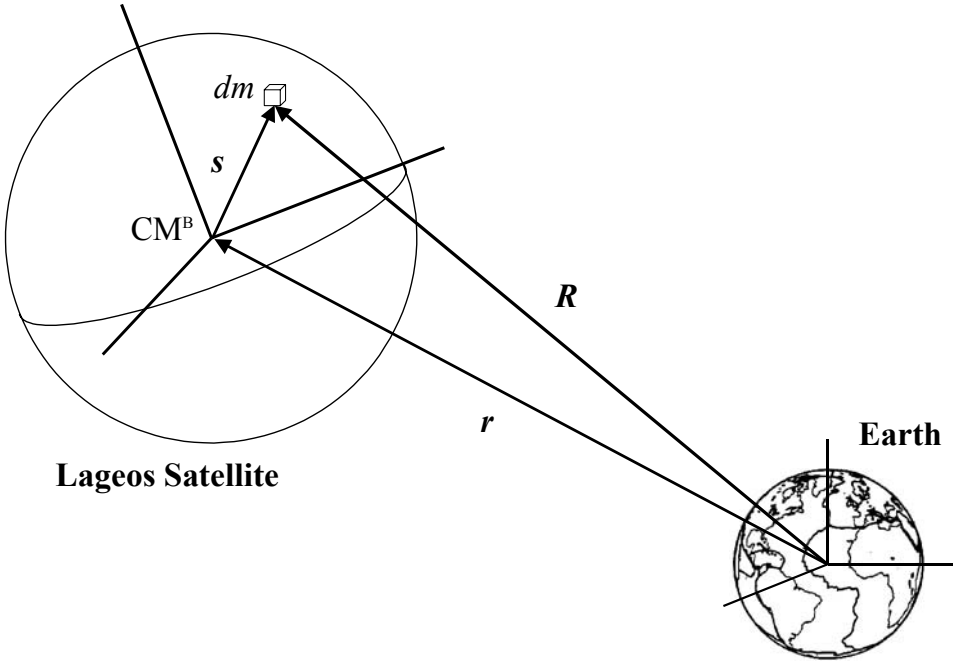

Figure 3.4 Schematic of the "simple" gravitational torque problem for Lageos

Reference coordinate axes representing the Body and ECI frames are exhibited. The satellite mass element *dm* is located in the body and ECI frames respectively by $\boldsymbol{s}$ and $\boldsymbol{R}$. The ECI position of Lageos' center of mass (CM^B) is given by $\boldsymbol{r}$.

$$\boldsymbol{N}_g = -\mu_e \int_B \frac{\boldsymbol{s}\times\boldsymbol{r}}{|\boldsymbol{r}+\boldsymbol{s}|^3} dm\,. \tag{32}$$

The solution to (32) is obtained using a *spherical harmonic* expansion for the satellite body; although, we halt our development before obtaining the traditional spherical harmonic form. First, we expand

$$|\boldsymbol{r}+\boldsymbol{s}|^2 = (\boldsymbol{r}+\boldsymbol{s})\cdot(\boldsymbol{r}+\boldsymbol{s}) = r^2\left[1-\left(\frac{s}{r}\right)\left(2\gamma-\left(\frac{s}{r}\right)\right)\right] \tag{33}$$

with γ the *direction cosine* between $-\boldsymbol{r}$ and $\boldsymbol{s}$, $\gamma = (-\hat{\boldsymbol{r}}) \cdot \hat{\boldsymbol{s}}$.[4] Since $s << r$, we use a binomial series to obtain

$$\frac{1}{|\boldsymbol{r}+\boldsymbol{s}|} = \frac{1}{r}\left[1 + \gamma\left(\frac{s}{r}\right) + \frac{1}{2}(3\gamma^2 - 1)\left(\frac{s}{r}\right)^2 + \mathrm{O}\left(\left(\frac{s}{r}\right)^3\right)\right]. \tag{34}$$

This is, in fact, an expansion of Legendre polynomials in the form

$$\frac{1}{|\boldsymbol{r}+\boldsymbol{s}|} = \frac{1}{r}\sum_{n=0}^{\infty}\left(\frac{s}{r}\right)^n P_n(\gamma). \tag{35}$$

Returning attention to (32), we can reverse the cross product and pull the constant $\boldsymbol{r}$ out of the integral. Then, using the results above, we find[5]

$$\boldsymbol{N}_g = \frac{\mu_e}{r^3}\boldsymbol{r} \times \left[\int_B \boldsymbol{s}\, dm + 3\int_B \gamma \frac{s}{r}\boldsymbol{s}\, dm + \frac{3}{2}\int_B (5\gamma^2 - 1)\left(\frac{s}{r}\right)^2 \boldsymbol{s}\, dm + \ldots\right]. \tag{36}$$

The first integral in (36) vanishes by definition of the center of mass. The second integral is our "first order" solution for the gravitational torques. However, before simplifying this solution, we can make a more general observation. Each of the terms of the series (36) involves polynomials in γ, and so they reduce to integrals of the form

$$k\int_B \gamma^n \left(\frac{s}{r}\right)^n \boldsymbol{s}\, dm = -\frac{k}{r^{2n}}\int_B (\boldsymbol{r}\cdot\boldsymbol{s})^n \boldsymbol{s}\, dm. \tag{37}$$

---

[4] We use $-\boldsymbol{r}$ to conform with convention. The direction cosine for spacecraft positions is taken with the "down," i.e., Earth pointing direction.

[5] This is an expansion in terms of Legendre polynomial first derivatives.

Appealing to the symmetries of Lageos, we claim these vanish for all even values of n. We will skip the details but sketch the argument. Using cylindrical coordinates (ρ, θ, z), write $\boldsymbol{r} \cdot \boldsymbol{s} = r_1\rho\cos\theta + r_2\rho\sin\theta + r_3 z$ and note the r_i are constant. Now observe each term in each vector component of $(\boldsymbol{r}\cdot\boldsymbol{s})^n\boldsymbol{s}$ includes products of the form $z^u\cos^v\theta\,\sin^w\theta$ for integers u, v, or w. Since n is even, only one of u, v, or w can be odd. If u is odd, the z integral will vanish due to the reflective symmetry of Lageos. On the other hand, if v or w are odd, Lageos' rotational symmetry eliminates the θ integral. This completes the argument.

For Lageos, the truncation error in keeping just the first non-vanishing term in (36) is negligible. After integration, the series contains only even powers of R_L/r where R_L is defined as the radius of the Lageos spacecraft. Using appropriate values, this results in a relative error for the primary solution of $O(10^{-15})$. Clearly, one cannot hope to do much better!

We now proceed to put the solution in a more convenient form. First, write

$$\boldsymbol{N}_g = \frac{3\mu_e}{r^3}\boldsymbol{r}\times\int_B \gamma\frac{s}{r}\boldsymbol{s}\,dm = -\frac{3\mu_E}{r^5}\boldsymbol{r}\times\left(\int_B \boldsymbol{ss}\,dm\right)\cdot\boldsymbol{r} \qquad 38$$

where we have expanded γ and extracted the constant $\boldsymbol{r}$ from inside the integral. Next, we can add a zero inside the integral in the form of $\boldsymbol{0} = \boldsymbol{r}\times(s^2\boldsymbol{I})\cdot\boldsymbol{r}$ so that

$$\boldsymbol{N}_g = -\frac{3\mu_e}{r^5}\boldsymbol{r}\times\left(-\int_B (s^2\boldsymbol{I} - \boldsymbol{ss})\,dm\right)\cdot\boldsymbol{r} = \frac{3\mu_E}{r^3}\hat{\boldsymbol{r}}\times\boldsymbol{I}\cdot\hat{\boldsymbol{r}}, \qquad 39$$

using (8) to achieve the final form. Now, let ρ_i be the body frame components of $-\hat{\boldsymbol{r}}$, i.e., $\rho_i = -\hat{\boldsymbol{r}} \cdot \boldsymbol{e}_i^{\mathrm{B}}$. Since $\boldsymbol{I}$ is diagonal (recall (10)), we finally simplify (39) to

$$\boldsymbol{N}_g = \frac{3\mu_e}{r^3}(I_3 - I_1)\begin{bmatrix} \rho_2\rho_3 \\ -\rho_1\rho_3 \\ 0 \end{bmatrix} \qquad 40$$

This result is a remarkably simple and elegant expression for the gravitational torques and is the form used in the H&W model. There is very little room (or reason) for improvement with this result from a satellite properties standpoint. Therefore, this also is the basis of the gravitational torque module for our current efforts.

One important observation about (40) is that the torque's magnitude is proportional to the difference in the principal moments. This difference is small, about 3% of their magnitude, suggesting a small relative change (or error!) in the principal moments could have significant impact on the torques. We defer the evidence and further discussion of this effect to Chapter 4.

3.4.2 Higher Order Corrections

While the higher order terms from the satellite spherical harmonics were inconsequential, there are other considerations. The two assumptions that allowed the reduction of (30) to a manageable form are now examined further.

To this point, we have considered only the gravity gradient across the satellite due to the Earth. In doing so, we made an implicit appeal to the μ_n / R_n^3 scaling of the problem. Still, it might be expected that the Sun (due to its mass) and the Moon (due to its relative

proximity) make non-trivial contributions to the net effect. However, a quick calculation verifies the effects are negligible from a practical standpoint. For Lageos' orbit, both the Sun and the Moon provide a relative contribution of $O(10^{-7})$ as compared to the Earth.

The second simplification is not as easily justified. The Earth is not a spherically symmetric mass. Rather, it has a complex mass distribution so that the gravitational field deviates from that of the point-mass approximation.

As with the satellite body above, the Earth's departure from sphericity is expressed in terms of spherical harmonics. The development is identical in reverse. The satellite is initially considered as a point mass, and the integration performed over the volume of the Earth. Typically, it is expressed in terms of the *Geopotential*

$$U = -G\int_e \frac{1}{R}\, dm_e \tag{41}$$

from which torques can be derived (see e.g., Hughes [1986] or Gill & Montenbruck [2000]. $1/R$ has the Legendre polynomial expansion of (35) where $\boldsymbol{s}$ now is the ECI position of the Earth mass element dm_e. Rather than keeping the expression in terms of the direction cosine γ, however, the expansion uses latitudinal and longitudinal angles, λ and ϕ respectively. Leaving the most general form in the realm of geodesy, we note that if the Earth is considered to have rotational symmetry (e.g. a spheroid), the result is an expansion in terms of *zonal harmonics*

$$U = -\frac{\mu_e}{r}\left[1 - \sum_{i=2}^{\infty} J_i \left(\frac{R_e}{r}\right)^i P_i(\sin\lambda)\right] \tag{42}$$

where R_e is the equatorial radius of the Earth, and the zonal coefficients J_i are empirically determined.

For Lageos, $R_e/r \approx ½$ so the terms do not diminish as rapidly as the corresponding terms for the satellite body. Fortunately, the first of the zonal coefficients, J_2, is dominant by better than 2 orders of magnitude, so it is sufficient to examine the impact of just the first term in (42). The development retraces the footsteps of the previous section now with an additional term. It is exceedingly messy; we refer to Hughes [1986] for the details and merely present the results:

$$\Delta \boldsymbol{N}_g = \frac{3\mu_e J_2 R_e^2}{2r^5}(I_3 - I_1)\begin{bmatrix} 5(1-7\sin^2\lambda_c)\rho_2\rho_3 - 10\sin\lambda_c(\rho_2 T_{33}^{\mathrm{EB}} + \rho_3 T_{23}^{\mathrm{EB}}) - 2T_{23}^{\mathrm{EB}}T_{33}^{\mathrm{EB}} \\ -\left[5(1-7\sin^2\lambda_c)\rho_1\rho_3 - 10\sin\lambda_c(\rho_1 T_{33}^{\mathrm{EB}} + \rho_3 T_{13}^{\mathrm{EB}}) - 2T_{13}^{\mathrm{EB}}T_{33}^{\mathrm{EB}}\right] \\ 0 \end{bmatrix} \quad 43$$

where the ρ_I are as in (40), λ_c is the latitude (declination) of CM^{B}, and the T_{i3}^{EB} are the components of the ECI z-axis in the body frame (see (2)).

Implementation

Equation 43 shows that the correction term for the spacecraft torque due to the J_2 zonal harmonic scales to $J_2 R_e^2/r^2 \approx 3\times10^{-4}$ relative to the primary term, so it is still quite small. Nevertheless, we believe it is worth including, particularly for high accuracy

Table 3.3 Parameter values for the gravitational torque model

The Geophysical constants are taken from the IERS Conventions document [J]; the satellite moments were stated earlier.

μ_e	3.986004418×10^{20} cm^{3}s^{-2}
J_2	1.0826359×10^{-3}
R_e	6.3781366×10^{9} cm
I_1	1.271×10^{8} g cm^2
I_3	1.314×10^{8} g cm^2

work as the model evolves. Accordingly, we have made this correction term available in the model as a selectable option.

Taking advantage of efficiencies, the implementation of the gravitational torque component in the W02 model is summarized as:

1. Given $\boldsymbol{r}$; Constants & Parameters: μ_e, I_1, I_3, J_2
2. Compute $(\rho_1,\ \rho_2,\ \rho_3) = \boldsymbol{T}^{\mathrm{EB}} \cdot (-\hat{\boldsymbol{r}})$
3. Set $\beta = \dfrac{3\mu_e (I_3 - I_1)}{r^3}$; $\quad \beta_1 = \rho_3 \beta$
4. Evaluate $\boldsymbol{N}_g = (\beta_1 \rho_2,\ -\beta_1 \rho_1,\ 0)$

   If higher order correction is selected . . .

5. Set $\beta_2 = \dfrac{J_2 R_e^2}{2r^2}\beta$; $\quad a_1 = 5\left[1 - 7(\hat{r}_3^{\mathrm{E}})^2\right]\rho_3$; $\quad a_2 = -10\hat{r}_3^{\mathrm{E}}$; $\quad a_3 = -2T_{33}^{\mathrm{EB}}$ where we note that $\sin\lambda_c = \hat{r}_3^{\mathrm{E}}$.
6. Add correction term $\boldsymbol{N}_g = \boldsymbol{N}_g + \beta_2 \begin{bmatrix} a_1\rho_2 + a_2(\rho_2 T_{33}^{\mathrm{EB}} + \rho_3 T_{23}^{\mathrm{EB}}) + a_3 T_{23}^{\mathrm{EB}} \\ -\left[a_1\rho_1 + a_2(\rho_1 T_{33}^{\mathrm{EB}} + \rho_3 T_{13}^{\mathrm{EB}}) + a_3 T_{13}^{\mathrm{EB}}\right] \\ 0 \end{bmatrix}$

Table 3.3 shows the values we have used in the model for the equations above.

A final remark before we move on. In most cases, the additional computational cost and complexity of incorporating the J_2 term into the magnetic torque calculation is probably not rewarded with a meaningful change in spin state behavior. Possibly, such effects can be mostly accounted for with much less cost; referring to (43), we see a way to proceed.

Averaging the latitudinal dependence over an orbit, the center term will vanish (in approximation), and the first term evaluates to a coefficient an order of magnitude larger than the final term. It is reasonable to assume the two corresponding component products $\rho_i\rho_3$ and $T_{i3}^{\mathrm{EB}}T_{33}^{\mathrm{EB}}$ have roughly the same magnitude. Therefore, the latter can be discarded in approximation due to the larger coefficient of the former, and we obtain

$$\Delta \boldsymbol{N}_g \approx \frac{3\mu_e}{r^3}(I_3 - I_1)\begin{bmatrix} \rho_2\rho_3 \\ -\rho_1\rho_3 \\ 0 \end{bmatrix} \cdot \frac{J_2 R_e^2}{2r^2} 5\,(1 - 7\overline{\sin^2 \lambda_c}) = \boldsymbol{N}_g \cdot \frac{J_2 R_e^2}{2r^2} 5\,(1 - 7\overline{\sin^2 \lambda_c}). \qquad 44$$

Evaluating, we find $\Delta \boldsymbol{N}_g \approx -1\times 10^{-3} \cdot \boldsymbol{N}_g$. Therefore, to incorporate the zonal term effect as an approximation, we are led to consider a parameterization of the form

$$\Delta \boldsymbol{N}_g \approx -\varepsilon \cdot \boldsymbol{N}_g \qquad 45$$

where ε is a parameter on the order of 10^{-3}.

This approach is available as a selectable option in the model. However, none of the analysis we performed made use of this feature.

3.5 Magnetic Torque Model

Modeling the effects of the spacecraft's interaction with the Earth's magnetic field is by far the most interesting and challenging aspect of this work. Unlike the situation for gravitational torques where the problem is well defined (and thoroughly investigated), numerous uncertainties and a dearth of resources pervade the magnetic torque problem. Even if we had a good understanding of the electric properties of the satellite (we don't),

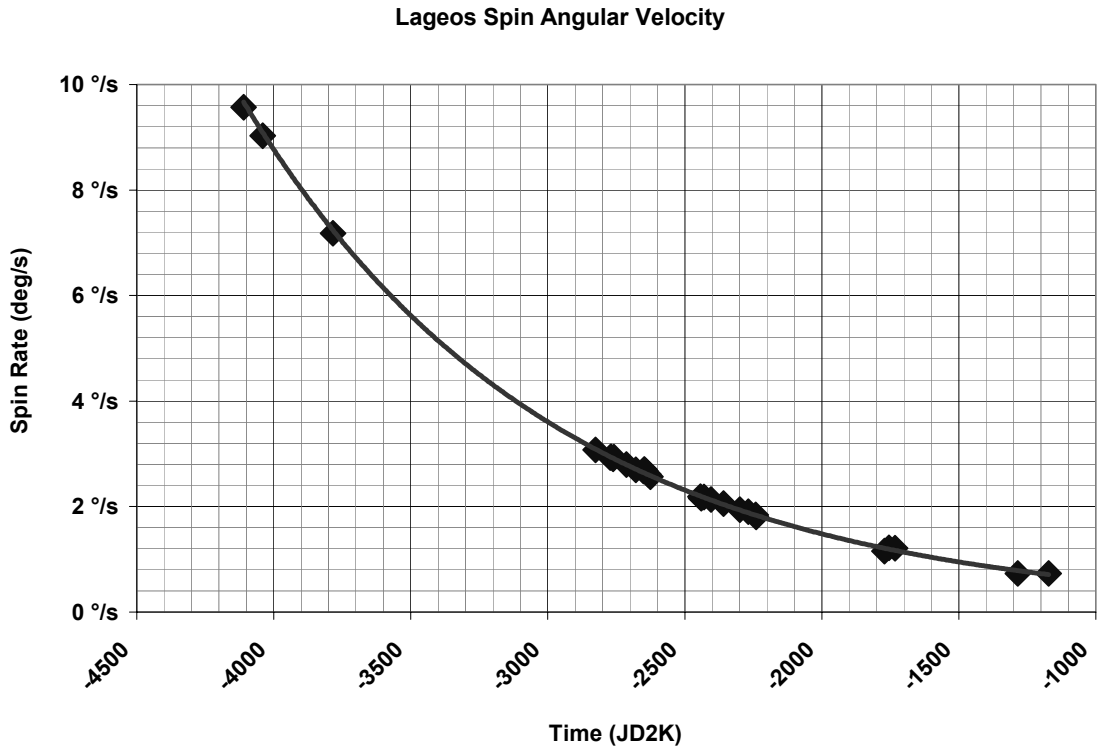

Figure 3.5 Historical values of Lageos' body spin rate

Empirical spin rate values for Lageos as determined by Avizonis [1997] are shown. The data is overlaid by a best-fit exponential decay curve that suggests a decay half-life of approximately 780 JD. Reproducing this behavior is a primary concern in choosing parameter values for the magnetic torque model

there still remains a problem that is scarcely mentioned in the literature, even in its most general abstraction. Before we get too far ahead of ourselves, though, we begin with a summary of the situation.

Magnetic torques arise as a result of the interaction of the metallic body of the spacecraft with the magnetic field surrounding the Earth. The effect is a consequence of the motion of the satellite with respect to the field—primarily spin early in life but later also the orbital motion. This results in the generation of *eddy currents* that induce a

dipole, which then reacts with the field to cause torque on the satellite. The eddy currents also produce heat so that energy is dissipated, i.e., the spin rate decays (Figure 3.5). Chobotov [1991] provides the torque expressions for these two interactions:

$$\boldsymbol{N}_m = \boldsymbol{M} \times \boldsymbol{B} \tag{46}$$

$$\boldsymbol{N}_{ec} = k(\boldsymbol{\omega} \times \boldsymbol{B}) \times \boldsymbol{B} \tag{47}$$

where $\boldsymbol{M}$ is the net magnetic moment of the satellite, $\boldsymbol{B}$ is the magnetic field, and k is a constant that depends on the geometric and electric properties of the satellite. If the rotation of the satellite is properly accounted for in the determination of the magnetic moment, $\boldsymbol{M}$, then (46) implicitly includes the eddy current torque of (47).

From equation (46) it is clear that the problem of magnetic torques involves two parts, one concerning the spacecraft—determining the magnetic moment $\boldsymbol{M}$, and one a purely environmental issue—determining the magnetic field $\boldsymbol{B}$. While some uncertainties exist with both aspects, the former is where most of our issues lie. We tackle the magnetic field model first and then proceed to the intricacies of the spacecraft model.

3.5.1 Earth Magnetic Field

The magnetic field near the Earth is the combined effect of three sources—the *Main Field* due to currents in the outer core, the *Crustal Field* resulting from magnetized rock near the Earth's surface, and the *External Field* generated by charged particles in the space environment [L]. There is therefore an inherent ambiguity in references to the "Earth magnetic field" because it is not always clear which effects are included. Typical

for problems of the sort discussed here is to treat the net of internal effects with a single field model and ignore external effects altogether. We proceed in this manner as well, noting the external field contributes less than 1% of the net field at Lageos' altitude.[6] It is almost convention to identify the net field due to only internal sources as the *geomagnetic field*; we make it convention here.

The general approach for the geomagnetic field model then proceeds much like the gravitational case above. In particular, the field can be expressed in terms of the gradient of a scalar potential that satisfies a differential equation. This leads to an integral of the type in (41) (see, e.g., Jackson [1975]). We have already seen this to have solutions in terms of spherical harmonics. Therefore, in complete agreement with the gravitational case, high precision geomagnetic field models are expressed as a spherical harmonic series. The corresponding empirically determined series coefficients are called *Guass Coefficients* (Campbell, [1996]). Unlike the gravitational case, however, the magnetic field is dynamic so new sets of Gauss Coefficients are required on a regular basis to keep up with the time-dependent system. We will return to this thought shortly, but first investigate the simpler approaches employed in the H&W model.

---

[6] The 1% value for the external field contribution is based on empirical analysis we performed using the web-based Tsyganenko T96_01 External Field Model at [M]. It is consistent with the generally reported values of < 1-2% for satellites in low and medium orbits.

Simple Magnetic Field Model

Some of the basic features of the Earth's magnetic field are familiar in everyday life—a magnetic north pole located in the general vicinity of the geographic north pole and a "mysterious" force that causes compass needles to point toward it. Extrapolating a bit, we find the magnetic field near the Earth behaves a lot like that of the simple dipole introduced in an elementary physics class. Indeed, about 90% of the geomagnetic field is explained by a dipole approximation with an axis inclined by about 11° from the Earth's spin axis (Chobotov, [1991]). Accordingly, this type of approximation is frequently used to model the magnetic field experienced by orbiting satellites.

The dipole field at the satellite is given by (e.g., Jackson [1975])

$$\boldsymbol{B} = \frac{3\boldsymbol{r}(\boldsymbol{r}\cdot\boldsymbol{M}_e) - r^2\boldsymbol{M}_e}{r^5} \qquad 48$$

where $\boldsymbol{M}_e$ is the geomagnetic *dipole moment* vector. Specifically, $\boldsymbol{M}_e$ lies along the dipole axis in the magnetic north direction; its magnitude determines the strength of the field. A nominal value for M_e is 7.8×10^{25} guass cm^3, though recall the field is dynamic so this value is non-constant in time.

Both the initial version of the H&W model and the Williams [1997] update use a dipole approximation as in (48). The first implementation (i.e., Habib et al [1994]) resolved the dipole along the Earth's spin axis, removing time dependence due to the Earth's rotation. This gave the general spin propagation problem complete rotational independence and so allowed a number of other simplifications as well (e.g., ignoring

precession of orbit plane). Given the angle between the "true" dipole axis and the Earth's spin axis is only 11°, this is certainly reasonable, particularly in light of the purpose of their efforts.

Nevertheless, we felt it was important to include the rotating dipole in the 1997 revision of the model. Unfortunately, we introduced the pole with a minor error, though we take comfort that we are not alone (Campbell, [1996]). The geographic position of the magnetic dipole north pole is quite distinct from the "magnetic north" that attracts compass needles. The reason is that the dipole approximation is actually just the first term of the spherical harmonic expansion for the geomagnetic field. As such, dipole pole positions are determined by the Guass Coefficients, which, in turn, represent the 'best fit' of the spherical harmonic model to the field everywhere, not just at one point. Magnetic

Table 3.4 Recent-history north pole locations for the geomagnetic field dipole approximation

The first order approximation to the Earth's magnetic field is a centered dipole with axis inclined relative to the Earth's rotation axis. The geographic pole locations wander in time due to the dynamic nature of the magnetic field. The location of the northern hemisphere pole is shown for a period spanning Lageos' orbital life. Also shown is the corresponding magnetic dipole moment. The data is computed from coefficients in the IGRF model. The 2005 data are based on a linear extrapolation of model parameters.

Year	Dipole Moment $\times 10^{25}$ gauss cm^3	Latitude	Longitude
1975	7.939	78.69°	289.53°
1980	7.907	78.81°	289.24°
1985	7.871	78.97°	289.10°
1990	7.841	79.13°	288.89°
1995	7.812	79.30°	288.59°
2000	7.788	79.54°	288.43°
2005	7.764	79.75°	288.27°

north, on the other hand, is the Earth surface location where the magnetic field lines happen to be the most vertical. Ideally, the two poles would be the same but, in fact, they can differ by as much as 30° longitude.

The impact of this error on earlier data sets was minor, with measurable effects showing up only for very long integration intervals. The correct dipole moment position and magnitude parameters are given in Table 3.4. However, the use of the dipole approximation as a stand-alone option has been replaced in our new version by the method of the following section.

International Geomagnetic Reference Field Model

Consistent with our "bottom-up" theme, we are not inclined to be satisfied with a 90% accurate approximation to the Earth's magnetic field, particularly in light of the scale of the corrections we suggested in previous sections. Given the similarity between the general representations of the gravitational and magnetic fields, we might also expect the high accuracy improvement for the magnetic model to proceed analogous to the gravitational model. However, because the geomagnetic field is computed independent of spacecraft properties, things are not quite so messy here. All we require is a straight evaluation of the spherical harmonic series.

Fortunately, this work is already done for us. The International Association of Geomagnetism (IAG) publishes the *International Geomagnetic Reference Field* (IGRF) with Guass Coefficients up to the 10^{th} term of the spherical harmonic series. The

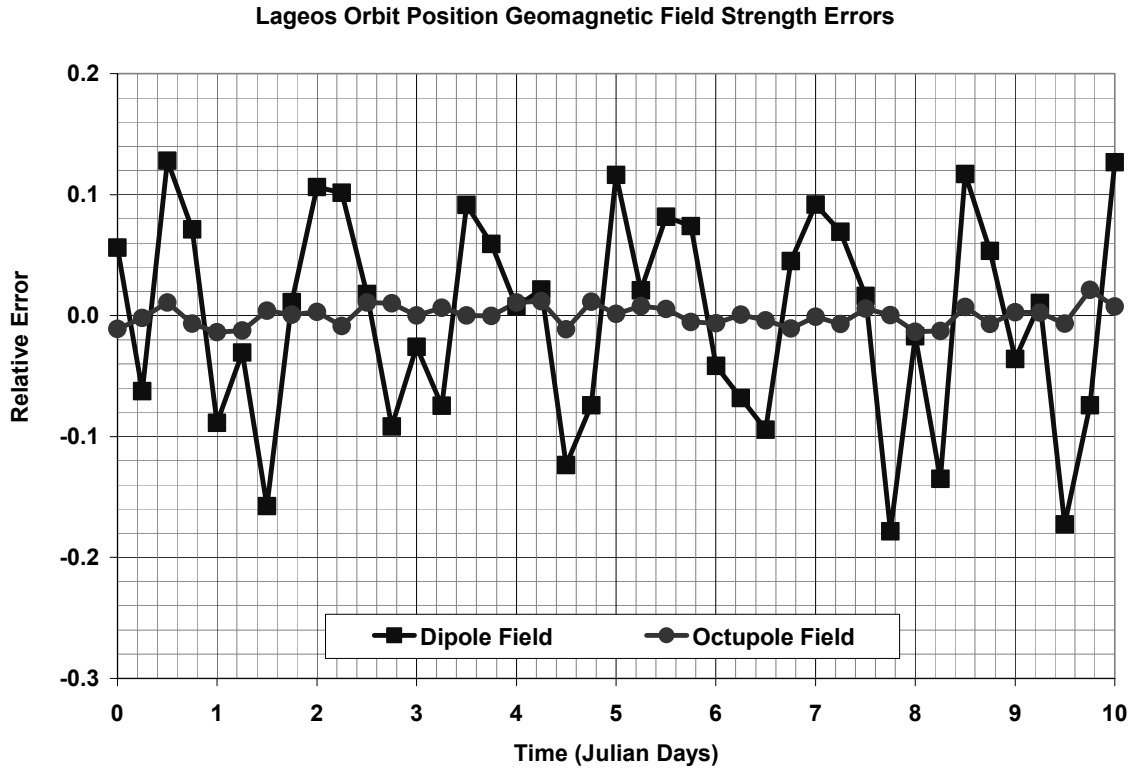

Figure 3.6 Sample dipole and octupole magnetic field strength errors at Lageos orbit positions

Dipole and octupole geomagnetic field approximations were computed at Lageos' orbit positions for a ten-day time interval using a course sampling interval (¼ day). The plot shows the relative errors of the approximations as compared to the full (10-stage) IGRF model. The octupole expansion represents a reasonable balance between accuracy and computational cost.

coefficients are updated every 5-years and the IAG recommends a discrete year-to-year linear interpolation to adjust coefficients between the published data sets [P]. The IGRF includes source code and so we have incorporated the model directly into our own (we use the *GEOPACK* adaptation by Tsyganenko [i]). However, it is excessive (and computationally expensive) to use the full 10-term expansion. We have therefore included an option to define the number of terms (n = 1 to 10) to be used. Empirical

observation has led us to favor the *octupole* expansion $n = 3$ as a balance between accuracy and computational cost (Figure 3.6). The additional advantage of this implementation is the dipole approximation is still available ($n = 1$) for less precise work.

3.5.2 Probing the Satellite Magnetic Torque Problem

We return our gaze to (46)

$$\boldsymbol{N}_m = \boldsymbol{M} \times \boldsymbol{B} \,. \tag{46}$$

Unfortunately, the situation for $\boldsymbol{M}$ is not as easily resolved as for $\boldsymbol{B}$. The complexities involved with deriving an appropriate model for the satellite are many and a number of significant abstractions will have to be made in order to proceed.

Recall the mechanism for magnetic torque is the generation of eddy currents in the satellite body. So the question is, how are such currents encouraged or inhibited by Lageos' structure? Were Lageos a homogeneous body with desirable geometry, the problem would at least be tractable; although, even then the situation is hardly straightforward. Of course, Lageos is not at all homogeneous, even if it is relatively simple. Therefore, we seek first to obtain a basic understanding of its structures as they apply to the magnetic torque problem. To summarize:

- The degree of electrical connectivity between the three major parts—the two hemispheres and the core—is uncertain. Bertotti & Iess [1991] have argued that oxidation on the aluminum surfaces makes it likely that these components are in poor electrical contact and so can be considered as three distinct conductors.

- The beryllium-copper core seems to be the best candidate for sustainable eddy-current generation because it is geometrically convex and materially homogeneous.
- Current generation in the shell is uncertain. The lack of electrical contact between the two hemispheres would seem to inhibit current generation as does the presence of the retroreflectors at the surface. Contrary to the latter, however, if the shell is sufficiently thick, there is no reason to believe currents can't exist inside the reflector layer. This is particularly true at low frequencies when the 'skin effect' (concentration of currents at the surface) is less likely. Clearly, the situation is mixed for the shell and it will be difficult to properly account for the effects.
- The shell may provide some degree of electromagnetic shielding to the field experienced by the core. Jackson [1975] shows that for a non-rotating spherical shell, shielding merely acts to dampen the magnitude of the field inside the shell—the direction remains unchanged. For rotation and/or variable fields, however, the situation is more complex.
- There are several different alloys associated with both 6061 aluminum and beryllium-copper with a range of electrical properties in each case. According to [Q], the range of effective conductivities for 6061 aluminum is 2.04×10^{17} to 2.37×10^{17} s^{-1} and the range for beryllium-copper is 0.88×10^{17} to 1.10×10^{17} s^{-1}. Other sources list values that sometimes fall outside of these ranges.

In addition to the uncertainties about the satellite discussed above, there are other complicating factors with regard to the basic problem. In particular, the magnetic moment induced on the satellite depends on the motion of the magnetic field relative to the spacecraft. For a simple periodic field, Landau & Lifshitz [1984] (henceforth L&L) show it is possible to provide an expression for the induced magnetic moment for relatively simple media. We make use of these results in the sequel because they represent the closest approximation in the literature to the problem at hand but they fall short of ideal. In the case of Lageos, the field varies on several different periodic timescales—the spin of the satellite, the orbital motion, and the rotation of the Earth—so a generalization of the L&L result to a multiple frequency case is desired. Moreover, the 'simple' media considered by L&L is at best a loose approximation to the conductors aboard Lageos. Unfortunately, a rigorous extension of the L&L results to accommodate these deficiencies is far from trivial and ultimately unpractical.

Therefore, some very basic decisions must be made about how to represent Lageos in the magnetic torque module. The H&W model chose to use the L&L result (single frequency, homogeneous sphere) directly as a single approximation for the entire response of the satellite. Obviously, based on the preceding arguments, such an approach necessarily omits dynamics important to accurate prediction of the satellite spin state. Nevertheless, the approach remains as the central element even for the present work (W02 model), though we have found ways to improve upon the basic idea and capture additional effects.

3.5.3 Primary Magnetic Torque Model

Two observations help justify the direct application of the L&L solution to determine the magnetic torque affecting Lageos. First, for frequencies on decidedly different scales, only the highest plays a significant role in the torque problem. For Lageos, the spin frequency has been dominant, allowing other timescales to be neglected in the earlier modeling work. Unfortunately, however, the spin rate is now rapidly approaching the orbital frequency[7] due to the ongoing decay. Indeed, perhaps extending as far back as a decade, the relative scale of the orbital frequency has been large enough that the corresponding effects contribute non-trivially to the results. Still, as a first order approximation, only the spin frequency is considered.

Second, the current inhibiting features and higher effective conductivity of the shell make its contribution to the magnetic torque comparatively small relative to the core. Therefore, only the core need be considered in the model and it is reasonably approximated by a sphere. Thus, the problem reduces to that of a homogeneous conducting sphere rotating uniformly in a static external field. We now summarize the L&L solution for this problem.

---

[7] The field frequency due to the orbital motion is actually twice the orbit frequency because of the pseudo-symmetries of the magnetic field. It is therefore a bit of an inaccuracy to speak of the "orbit frequency" but it is more convenient to do so and satisfies the general context. We will return to the point in more detail later. In the mean time one may substitute "twice the orbit frequency" every time the orbit frequency is mentioned. That established, note that the present day spin frequency of Lageos differs from this 2x orbit frequency by little more than a factor of two.

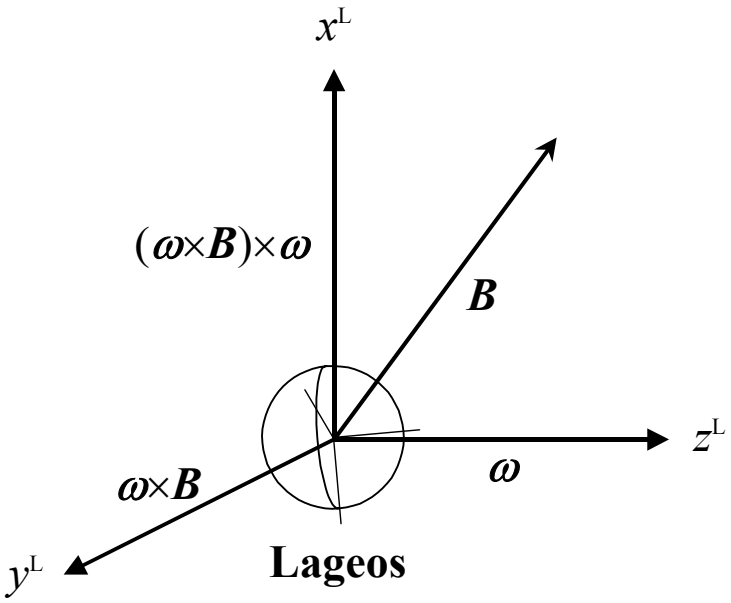

Figure 3.7 Schematic of the Landau-Lifshitz reference frame

The Landau-Lifshitz frame is constructed so that $\boldsymbol{\omega}$ and $\boldsymbol{B}$ lie in the *x*-*z* plane with $\boldsymbol{\omega}$ along the *z*-axis. A notional representation of the Lageos satellite is shown for illustration purposes.

Landau-Lifshitz Coordinate Frame

The principle results from L&L are expressions for the *coefficients of magnetization* of the sphere and the *Landau-Lifshitz reference frame* (LL) in which these can be used to compute the magnetic moment of the sphere, $\boldsymbol{M}$, and the corresponding torques. The LL frame is constructed so that the field, $\boldsymbol{B}^L$, has no *y*-component and the angular velocity, $\boldsymbol{\omega}$, is along the *z* axis (Figure 3.7). This is accomplished by defining the LL basis vectors as

$$\begin{aligned}(\boldsymbol{e}_1^L)^B &= (\boldsymbol{\omega}^B \times \boldsymbol{B}^B)\times\boldsymbol{\omega}^B \\ (\boldsymbol{e}_2^L)^B &= \boldsymbol{\omega}^B \times \boldsymbol{B}^B \\ (\boldsymbol{e}_3^L)^B &= \boldsymbol{\omega}^B\end{aligned} \qquad 49$$

where the scaling to unit magnitude is implicitly understood. We have expressed the result in relation to the body frame as a convenience with an eye toward implementation. Accordingly, we define the transformation

$$\boldsymbol{T}^{\mathrm{BL}} = \begin{pmatrix} (\boldsymbol{e}_1^{\mathrm{L}})^{\mathrm{B}} \\ (\boldsymbol{e}_2^{\mathrm{L}})^{\mathrm{B}} \\ (\boldsymbol{e}_3^{\mathrm{L}})^{\mathrm{B}} \end{pmatrix} \tag{50}$$

so that the LL components of $\boldsymbol{B}$ can be determined

$$\boldsymbol{B}^{\mathrm{L}} = \boldsymbol{T}^{\mathrm{BL}} \cdot \boldsymbol{B}^{\mathrm{B}} = \begin{pmatrix} \boldsymbol{B}^{\mathrm{B}} \cdot (\boldsymbol{e}_1^{\mathrm{L}})^{\mathrm{B}} \\ 0 \\ \boldsymbol{B}^{\mathrm{B}} \cdot (\boldsymbol{e}_3^{\mathrm{L}})^{\mathrm{B}} \end{pmatrix} \tag{51}$$

Magnetic Moment and Torque

In a moment we outline L&L's derivation of the *coefficients of magnetization*, α' and α'', for the sphere in this problem. Once these are known, the magnetic moment and torque are readily computed. By the construction of the LL frame, the field is not variable in the z direction and so does not induce a moment in that direction. The coefficients of magnetization give the components of the magnetic moment parallel to and perpendicular to the plane defined by $\boldsymbol{\omega}$ and $\boldsymbol{B}$ so that

$$\boldsymbol{M}^{\mathrm{L}} = \begin{pmatrix} V\alpha' B_1^{\mathrm{L}} \\ V\alpha'' B_1^{\mathrm{L}} \\ 0 \end{pmatrix}. \tag{52}$$

where V is the volume of the sphere, $V = \frac{4}{3}\pi a^3$. The torque is then immediately obtained from (46)

$$\boldsymbol{N}_m^{\mathrm{L}} = \boldsymbol{M}^{\mathrm{L}} \times \boldsymbol{B}^{\mathrm{L}} = \begin{pmatrix} V\alpha'' B_1^{\mathrm{L}} B_3^{\mathrm{L}} \\ -V\alpha' B_1^{\mathrm{L}} B_3^{\mathrm{L}} \\ -V\alpha'' (B_1^{\mathrm{L}})^2 \end{pmatrix}. \qquad 53$$

This result can then be transformed back to the body frame by $\boldsymbol{N}^{\mathrm{B}} = (\boldsymbol{T}^{\mathrm{BL}})' \cdot \boldsymbol{N}^{\mathrm{L}}$ and linearly superposed with $\boldsymbol{N}_g$ obtained earlier to determine the net torque on the satellite.

Coefficients of Magnetization

It would be convenient to simply state the results of L&L's development but it will be more useful to our later analysis to provide some of the details. The coefficients of magnetization are derived from the standard macroscopic field equations in magnetostatics and the general process is as described by Jackson [1975]. However, a reasonable solution presents itself only under the simplest of assumptions. Anticipating the notation, we define the *penetration depth*, δ, and a complex scalar constant, k, such that

$$\delta = \frac{c}{\sqrt{2\pi\sigma\omega}}, \qquad k^2 = \left(\frac{1+i}{\delta}\right)^2 = \frac{4\pi i\sigma\omega}{c^2} \qquad 54$$

where c is the speed of light and σ is the effective conductivity. The situation considered by L&L is for *quasi-static* magnetic fields, which they define, and we do not (for brevity). In short, it allows the field immediately outside the sphere to be described by the homogeneous equations

$$\begin{aligned} \nabla \cdot \boldsymbol{B}_x &= 0 \\ \nabla \times \boldsymbol{H}_x &= 0 \end{aligned} \qquad \boldsymbol{B}_x = \mu \boldsymbol{H}_x \tag{55}$$

where μ is the magnetic permeability. L&L argue that μ can be set to unity without loss of generality (it differs from unity only slightly for diamagnetic and paramagnetic bodies) and so it is dropped from the remainder of the discussion.

The equations (55) satisfy continuous boundary conditions at the surface (i.e., must equal the solution for the field inside the sphere) and must vanish at infinity. Since $\nabla \times \boldsymbol{H}_x = 0$, the field can be derived from a scalar potential, $\boldsymbol{H}_x = -\nabla\phi + \boldsymbol{B}$, which leads to an instance of LaPlace's Equation, $\nabla^2\phi = 0$. Also, ϕ depends linearly on $\boldsymbol{B}$, so L&L write $\phi = -V\alpha\boldsymbol{B} \cdot \nabla(1/r)$, where V is the volume of the sphere and α is a to be determined complex integration constant. Applying the gradient and simplifying, the field outside the sphere has the form

$$\boldsymbol{H}_x = \frac{V\alpha}{r^3}\left[3(\boldsymbol{B} \cdot \hat{\boldsymbol{r}})\hat{\boldsymbol{r}} - \boldsymbol{B}\right] + \boldsymbol{B} \tag{56}$$

Inside the sphere, the presence of current leads to the equations

$$\begin{aligned} \nabla \cdot \boldsymbol{B}_i &= 0 \\ \nabla \times (\nabla \times \boldsymbol{H}_i) &= -ik^2 \frac{\partial \boldsymbol{H}_i}{\partial t} \end{aligned} \qquad \boldsymbol{B}_i = \mu \boldsymbol{H}_i \tag{57}$$

These also satisfy the continuous boundary conditions and must be finite at the origin. In a coordinate frame in which the sphere is fixed, the uniform periodic external field has the form

$$\boldsymbol{B} = \boldsymbol{B}_o e^{-i\omega t} \tag{58}$$

where $\boldsymbol{B}_o$ is a constant complex vector. Separating variables (see e.g., Haberman [1987]), the time dependence in the curl equation in (57) is isolated and it is seen that $\boldsymbol{H}_i$ must also depend on time through the $e^{-i\omega t}$ factor. The generalized field equation in (57) becomes

$$\nabla \times (\nabla \times \boldsymbol{H}_i) = k^2 \boldsymbol{H}_i . \tag{59}$$

Since the gradient in (57) vanishes, the field can be derived from a vector, $\boldsymbol{H}_i = \nabla \times \boldsymbol{A}$. Symmetry arguments allow $\boldsymbol{A}$ to be written in terms of a scalar, $\boldsymbol{A} = \beta \nabla \times (f\boldsymbol{B})$, with f the spherically symmetric solution to $\nabla^2 f + k^2 f = 0$ (finite at the origin) and β an integration constant. This leads to

$$\boldsymbol{H}_i = \beta\left(\frac{f'}{r} + k^2 f\right)\boldsymbol{B} - \beta\left(\frac{3f'}{r} + k^2 f\right)(\boldsymbol{B}\cdot\hat{\boldsymbol{r}})\boldsymbol{B} \tag{60}$$

L&L claim the magnetic moment of the sphere is

$$\boldsymbol{M} = V\alpha\boldsymbol{B} . \tag{61}$$

and so α is the complex coefficient of magnetization. Applying the continuity conditions to (56) and (60) leads to a solution for α. Isolating the real and imaginary parts, the coefficients of magnetization of the sphere are

$$\alpha' = -\frac{3}{8\pi}\left[1 - \frac{3\delta}{2a}\frac{\sinh(2a/\delta) - \sin(2a/\delta)}{\cosh(2a/\delta) - \cos(2a/\delta)}\right] \tag{62}$$

$$\alpha'' = -\frac{9\delta^2}{16\pi a^2}\left[1 - \frac{a}{\delta}\frac{\sinh(2a/\delta) + \sin(2a/\delta)}{\cosh(2a/\delta) - \cos(2a/\delta)}\right] \tag{63}$$

where a is the radius of the sphere. In the low frequency limit, $2a/\delta$ is small and the magnetization coefficients can be replaced by approximations

$$\alpha' = -\frac{4\pi}{105}\frac{a^4\sigma^2\omega^2}{c^4} \quad \text{and} \quad \alpha'' = \frac{a^2\sigma\omega}{10c^2} \tag{64}$$

For double precision floating point accuracy, the low frequency limits can be employed when $2a/\delta < \sim 0.04$. However, as we noted earlier, Bertotti & Iess suggest (64) can be employed from the beginning of the Lageos mission.

Corollary Result: Implication for Effective Conductivity

One interesting implication of (64) is that materials with high effective conductivities play a larger relative role in the low frequency regime. This runs counter to some of the assumptions made in previous efforts that the high effective conductivity of the aluminum shell make it a non-player in the magnetic field induced dynamics. Yet, we can verify empirically that the modeled spin dynamics are more sensitive for higher values of σ.

To understand why, first observe $\alpha'' >> \alpha'$ for lower frequencies. It is therefore convenient to bypass α' in the following discussion. Define the parameter

$$\gamma = 2a/\delta = 2a\frac{\sqrt{2\pi\sigma\omega}}{c} \tag{65}$$

which is proportional to $\sqrt{\sigma}$. The expression for α'' is now

$$\alpha'' = -\frac{9}{4\pi\gamma^2}\left[1 - \frac{\gamma}{2}\frac{\sinh(\gamma)+\sin(\gamma)}{\cosh(\gamma)-\cos(\gamma)}\right]. \tag{66}$$

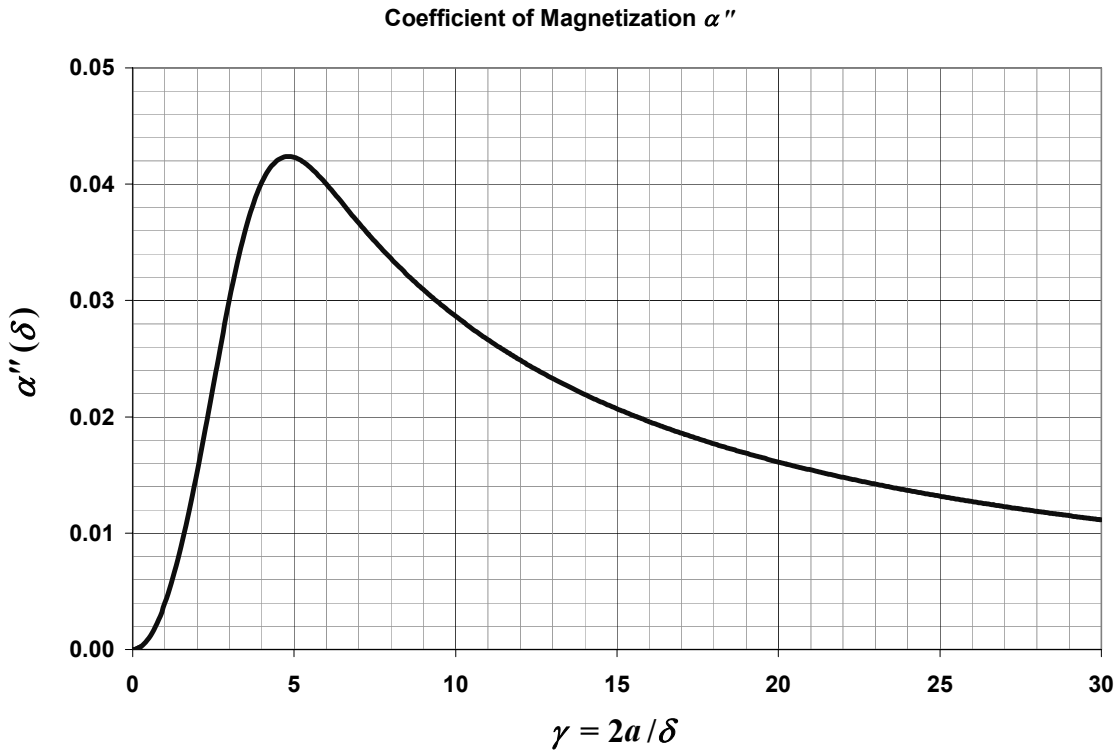

Figure 3.8 Global behavior of α'' as composite function of model parameters

The parameter γ is proportional to both $\sqrt{\omega}$ and $\sqrt{\sigma}$. In the *low frequency regime* (to the left of the peak), an increase in σ results in a larger coefficient of magnetization.

Figure 3.8 shows the behavior of (66) as a function of γ. For convenience, we define the region of the domain to the left of the peak as the *low frequency regime*. It can easily be verified that the historical spin frequencies of Lageos keep the behavior firmly rooted in the low frequency regime except near the beginning of the mission. It immediately follows that for any fixed frequency, increasing σ results in larger values of α'' and a greater impact on the behavior of the satellite. We conclude, therefore, that while there

are other reasons to potentially discount the shell's contribution to the magnetic torques, the material composition cannot be one of them.

Implementation

The preceding L&L result gives us a reasonable approach to the magnetic torque problem; it remains to specify the properties—effective conductivity and radius—of the *reference sphere*.[8] To do so, we recall the empirically measured decay in the spin rate of Lageos (Figure 3.5) and note that the magnetic torques are the only mechanism for dissipating that energy. Thus, at the very least, the reference sphere properties must be specified so that observed decay is matched by the model output.

Our approach is to choose a value for effective conductivity appropriate for the material composition of the core, $\sigma = 1.0\times10^{17}$ s^{-1} for example, and a-posteriori determine the radius to match model output to the spin rate data; a = 26.52 cm. Note the value is significantly larger than the dimensions of the cylindrical core. This confirms that the reference sphere should not be taken too literally as a satellite structure and hints at the limitation of the approximation. Nevertheless, the approach yields qualitatively accurate results and is a good foundation on which to base our later modifications.

[8] We call it a reference sphere to emphasize the point that it is not a satellite structure but an abstraction with primary purpose of reproducing the dynamical effects of the satellite.

The algorithm for the primary magnetic torque model proceeds as follows

1. Given $\boldsymbol{r}$; Constants and Parameters: σ, a, c
2. Obtain $\boldsymbol{B}$ (at $\boldsymbol{r}$) from field model and transform to the body frame, $\boldsymbol{B}^{\mathrm{B}} = \boldsymbol{T}^{\mathrm{EB}} \cdot \boldsymbol{B}$
3. Compute $\boldsymbol{\omega}$ from (4)
4. Compute the transformation to the LL frame using (49) and (50)
5. Transform magnetic field vector to LL frame $\boldsymbol{B}^{\mathrm{L}} = \boldsymbol{T}^{\mathrm{BL}} \cdot \boldsymbol{B}^{\mathrm{B}}$ (recall y component is zero by construction, therefore there is no need to explicitly compute it).
6. Pre-compute (one time at beginning of program):

$$k_1 = 2a\frac{\sqrt{2\pi\sigma}}{c}, \quad k_2 = \frac{9}{8\pi k_1}, \quad k_3 = -\frac{9}{4\pi k_1^2}, \quad k_4 = \frac{a^2\sigma}{10c^2}, \quad k_5 = \frac{4\pi}{105}\frac{a^4\sigma^2}{c^4}$$

7. Set $\beta_1 = k_1\sqrt{\omega}$ (note $\beta_1 = 2a/\delta$)
   a. If $\beta_1 < 0.04$, compute $\alpha' = k_5\omega^2$ and $\alpha'' = k_4\omega$
   b. Otherwise compute

$$\beta_1 = k_2/\sqrt{\omega}, \quad sh = \sinh\beta_1, \quad s = \sin\beta_1, \quad d = \cosh\beta_1 - \cos\beta_1$$

$$\alpha' = -\frac{3}{8\pi} + \beta_2\left(\frac{sh - s}{d}\right) \quad \text{and} \quad \alpha'' = \frac{k_3}{\omega} + \beta_2\left(\frac{sh + s}{d}\right)$$

8. Set $a_1 = VB_1^{\mathrm{L}}$ and $a_2 = a_1 B_3^{\mathrm{L}}$, then compute $\boldsymbol{N}_m^{\mathrm{L}} = (a_2\alpha'', -a_2\alpha', -a_1 B_1^{\mathrm{L}}\alpha'')$
9. Transform torque to body frame $\boldsymbol{N}^{\mathrm{B}} = (\boldsymbol{T}^{\mathrm{BL}})' \cdot \boldsymbol{N}^{\mathrm{L}}$

3.5.4 Analysis and Improvements

In the ideal, we would like an analytical solution for the general magnetic torque problem—multiple frequencies and multiple conductors with differing geometric and electric properties. We could, therefore, return to the statement of the problem and attempt to construct an entirely new approach from the ground up. However, for reasons already established, a strict analytical treatment is not only prohibitive but also perhaps impossible. Indeed, if the lack of literature is any indication, there does not seem to be much hope of a practical generalized solution that accommodates to the goal of numerical implementation.

It is therefore reasonable to maintain the perspective of the previous section. Specifically, rather than attempt to recreate the physical system in detail, we opt for a simpler reference approximation with the goal of reproducing the dynamical effects via the parameters of the model. Corrections that are rooted in, but not strictly based on the particulars of the physical system, can then be introduced. This approach provides the flexibility to adopt enhancements via parameterizations that are notionally valid in terms of the resulting dynamics but perhaps take some physical or mathematical liberties in the process.[9]

To seek out candidate improvements, we revisit and analyze the ways in which the L&L approach (single field frequency, single homogenous spherical conductor) violates

[9] The paradigm justifying this approach has already been established in the adoption of the reference sphere to represent the satellite.

the physical system. From there, we postulate possible corrections and investigate the implications. The three main points of interest are

- Core geometry
- Shell effects
- Multiple field frequencies

Magnetic shielding was also identified as a potential factor in the magnetic torque problem but explicit inclusion seems a needless refinement—any such effects are already accounted for within the other model parameters. Similarly, we can summarily dismiss the role of thermoelastic deformation on the magnetic torques. Even if the effect plays a role in the true physical system, any corresponding impacts are overshadowed by the abstractions already in place—it need not be considered separately.

Core Geometry

The cylindrical beryllium-copper core is approximated in the L&L model by an isotropic sphere. We argued earlier that this is a reasonable approximation and indeed, it is in many respects. For one thing, the cylindrical core is dimensionally similar to a sphere, being only slightly 'flattened' with a 31.76 cm diameter and 26.70 cm height. Also, the core possesses a uniform (homogeneous) material structure. Finally, the cylinder and sphere share many symmetry characteristics.

However, the two differ significantly in surface structure, a property extremely important to the magnetic torque problem because of the implications for current flow.

The directionally independent geometry of the sphere implies certain continuities of current flow, not only for any given fixed direction but also as a function of changing orientation. The same cannot be said for a cylinder, which has a non-smooth surface and so leads to interesting (and very difficult to quantify) effects for different orientations.

There is an immediate implication for the resulting magnetic torques. In particular, the magnetic moment of the cylinder will be attitude dependent, unlike the case for the sphere. Also, because of the complex boundaries of the cylinder, one can surmise additional resistance as compared to the sphere and hence greater energy loss due to Joule heating.

At this point we must admit that an elegant implementation of these ideas has been at our disposal all along. As it turns out, L&L also partly treated the case of a cylinder in their work. Unfortunately, the results are only valid for specific orientations of the cylinder relative to the magnetic field and only the coefficients of magnetization are computed—no discussion of the torque problem is presented. It is probably for these reasons that previous modeling efforts ignored L&L's cylinder solutions.[10]

To be specific, L&L provide the coefficients of magnetization for a conducting cylinder in a uniform periodic field in two specific cases: 1) magnetic field parallel to the cylinder axis (*longitudinal field*) and 2) magnetic field perpendicular to the cylinder axis (*transverse field*). One problem with the results is that the complex coefficient of

[10] Not only is this true of the H&W model, Bertotti & Iess chose the spherical solution from L&L over the cylindrical version as well.

magnetization does not separate nicely into real and imaginary parts as in (62) and (63) and so the solution is not particularly adaptable to numerical implementation. Moreover, no mention of the situation in between the two cases is given. And for good reason—it is not a tractable problem.

So, while the treatment of the cylinder by L&L is insufficient as a general solution in itself, some useful information can be gleaned. Two observations in particular guide the remedy we have implemented. First, the complex magnetization coefficient for the transverse field is exactly twice that of the longitudinal field. Second, in the low frequency limit, where separation of the complex coefficient is possible, the imaginary part of the transverse coefficient for the cylinder, α'', scales to 2 ½ times that of its spherical counterpart.[11] This seems a strong verification of the earlier deductions—attitude dependence and a magnification of the effects.

Extrapolating these ideas to the W02 model, two modifications are introduced. The first is a straight scaling parameter, κ, for the computation of the coefficients of magnetization and the second is the introduction of attitude dependence as a linear function of the direction cosine between $\boldsymbol{B}$ and $\boldsymbol{e}_3^{\mathrm{B}}$ (the body z axis is coincident with the longitudinal axis of the cylinder). The combined result is expressed as

$$\kappa' = \kappa \left(1 - \tfrac{1}{2} \left| \hat{\boldsymbol{B}} \cdot \boldsymbol{e}_3^{\mathrm{B}} \right| \right). \qquad 67$$

[11] The real part α' scales to better than 4 times its spherical analog but remains small relative to the imaginary part.

$$\begin{aligned} \widetilde{\alpha}' &= \kappa'\alpha' \\ \widetilde{\alpha}'' &= \kappa'\alpha'' \end{aligned} \qquad 68$$

Certainly, the true transitional behavior as a function of the field orientation from transverse to longitudinal is far more complex than the simple cosine dependence above. But this approach satisfies the goals of the approximation and closes the gap between the L&L sphere and the physical system.

Shell Effects

Heretofore it has been assumed that the shell makes little contribution to the magnetic torque on the satellite. Several reasons have been given—current inhibiting reflectors in the surface, a lack of electrical contact between the hemispheres, and a higher effective conductivity. We have already shown the last of these assumptions to be specious for lower frequencies and have also argued for the possibility of currents in a layer beneath the reflectors. The lack of electrical contact between the hemispheres is limiting to some extent but does not preclude currents altogether. In fact, it would seem to lead to a similar non-smooth surface situation we just discussed with the cylinder. All this suggests that the shell cannot be so easily dismissed in the magnetic torque problem.

Nevertheless, we now show that any such effects can largely be accounted for within the parameterization for the core geometry above. This is a qualitative argument so it will be convenient to treat the shell as a single entity; the hemisphere boundary effects complicate the issue but don't substantially change the conclusion. For the torque arising from the shell, we use (47)

$$\boldsymbol{N}_{ec} = k(\boldsymbol{\omega} \times \boldsymbol{B}) \times \boldsymbol{B} \tag{47}$$

and calculate the result in the LL frame. Since $\boldsymbol{\omega}^{\mathrm{L}}$ lies along the z-axis and $\boldsymbol{B}^{\mathrm{L}}$ has no y component, the solution is readily obtained

$$\boldsymbol{N}_{ec}^{\mathrm{L}} = \begin{pmatrix} k\omega B_1^{\mathrm{L}} B_3^{\mathrm{L}} \\ 0 \\ -k\omega (B_1^{\mathrm{L}})^2 \end{pmatrix}. \tag{69}$$

This can be linearly superposed with the result for the core (53) to obtain the net torque on the spacecraft

$$\boldsymbol{N}_m^{\mathrm{L}} = \boldsymbol{N}_{core}^{\mathrm{L}} + \boldsymbol{N}_{shell}^{\mathrm{L}} = \begin{pmatrix} (V\alpha'' + k\omega) B_1^{\mathrm{L}} B_3^{\mathrm{L}} \\ -V\alpha' B_1^{\mathrm{L}} B_3^{\mathrm{L}} \\ -(V\alpha'' + k\omega)(B_1^{\mathrm{L}})^2 \end{pmatrix}. \tag{70}$$

Now, since k is a function of the shell geometry and electric properties, then together, $k\omega \propto V_s \alpha_s''$ (more on this in a moment) where V_s is the volume of the shell and α_s'' is the imaginary part of the shell's coefficient of magnetization. If λ is a proportionality constant, then

$$k\omega = \lambda V_s \alpha_s'' = \lambda \frac{V_s \alpha_s''}{V\alpha''} V\alpha'' = \tilde{\lambda} V\alpha''. \tag{71}$$

and so (70) becomes

$$\boldsymbol{N}_m^{\mathrm{L}} = \boldsymbol{N}_{core}^{\mathrm{L}} + \boldsymbol{N}_{shell}^{\mathrm{L}} = \begin{pmatrix} V\alpha''(1+\tilde{\lambda}) B_1^{\mathrm{L}} B_3^{\mathrm{L}} \\ -V\alpha' B_1^{\mathrm{L}} B_3^{\mathrm{L}} \\ -V\alpha''(1+\tilde{\lambda})(B_1^{\mathrm{L}})^2 \end{pmatrix}. \tag{72}$$

where $\tilde{\lambda} \propto \frac{V_s}{V}\frac{\alpha_s''}{\alpha''}$. Thus, to a great degree, the additional torque due to the shell can be regarded as a scaling of the complex magnetization coefficient of the core, namely α''. Note that except perhaps very early in life, the spin frequency of Lageos is such that $\alpha'' >> \alpha'$ so the scaling in (72) can be reasonably absorbed by the scaling factor κ of the previous section. Therefore, no additional parameterization is required to account for the shell.

We now return to the assertion $k\omega \propto V_s\alpha_s''$. This is reasonable at face value, but further justification is warranted. First, observe that the shell possesses the same spherical symmetries of the core and so the L&L development proceeds similarly. The difficulty in pursuing the L&L approach for the shell directly, however, is due to the boundary condition on the interior shell surface. This boundary condition doesn't change the general nature of the overall solution, but prohibits a clean separation of the complex magnetization coefficient.

The system is described by two occurrences of (55)—one for the field on the outside of the shell and one for the cavity—and again by (59) for the shell itself. The respective solutions follow in the form of (56) and (60), specifically,

$$\boldsymbol{H}_x = \frac{V_s\alpha_s}{r^3}\left[3(\boldsymbol{B}\cdot\hat{\boldsymbol{r}})\hat{\boldsymbol{r}} - \boldsymbol{B}\right] + \boldsymbol{B} \qquad 73$$

$$\boldsymbol{H}_s = \left(\frac{f'}{r} + k^2 f\right)\boldsymbol{B} - \left(\frac{3f'}{r} + k^2 f\right)(\boldsymbol{B}\cdot\hat{\boldsymbol{r}})\boldsymbol{B} \qquad 74$$

$$\boldsymbol{H}_i = -\gamma\boldsymbol{B}\,. \qquad 75$$

We have changed the notation slightly—$\boldsymbol{H}_i$ now represents the field in the cavity and $\boldsymbol{H}_s$ is the field in the shell. Also note the k here is as in (54) and not the same as (69) – (72). The integration constants are α_s and γ along with two constants contained in f, the spherically symmetric solution to $\nabla^2 f + k^2 f = 0$ which differs from before in that the finite origin boundary condition is replaced by the continuity condition on the interior of the shell.

The remaining details are tedious and do not further inform this discussion so we proceed to the result. Letting b_1 the outer radius and b_2 the inner radius, the complex coefficient of magnetization for the shell can be written

$$\alpha_s = -\frac{b_1^3}{2V_s}\left[1 - \frac{3}{b_1^2 k^2} + \frac{3}{b_1 k}\cot kb_1 + \frac{3}{b_1 k}G(kb_1, kb_2)\right] \tag{76}$$

where G is a rather messy function of kb_1 and kb_2 but scales comparable to, if not smaller than, the other terms. Compare this to the complex coefficient of magnetization for a sphere of radius b_1

$$\alpha = -\frac{b_1^3}{2V}\left[1 - \frac{3}{b_1^2 k^2} + \frac{3}{b_1 k}\cot kb_1\right] \tag{77}$$

where V is the volume of the sphere. Therefore,

$$V_s\alpha_s = V_s\alpha_s\left[1 + \widetilde{G}(kb_1, kb_2)\right] \tag{78}$$

that is, $V_s\alpha_s \propto V\alpha$ for the shell in comparison to a sphere with the same outer radius. It also follows that the torque due to the shell will then resolve to a form similar to that of a

sphere as we claim above and that the resulting scaling can be accounted for within the parameter κ from the previous section.

Multiple Frequencies

The situation for multiple timescales in the magnetic torque problem is quite a bit more complicated. A key factor in the L&L development was the simple time dependence of $\boldsymbol{H}_i$ leading to a reduction to the form of (59). This allowed the somewhat straightforward procedure as presented by L&L. If multiple timescales are included, however, no such simplification can be made.

In the more general problem, not only are there multiple frequencies, but they also arise from different types of behavior. There is the spin of the body in the field (the case considered above) and the translational motion of the body through the field due both to orbital motion and the Earth's rotation. L&L argue that their results are equally valid for either case individually because the latter conforms to the former via a change of coordinates. Unfortunately, however, the combined effects force a restatement of the general problem so no direct application of the preceding results can be made, at least not from a rigorous standpoint.

To illustrate the point in more detail, consider only the two largest timescales—that of the satellite spin and that due to the orbital motion. Recall for the latter, the relevant frequency is twice the orbital frequency (referenced in the sequel by the prefix *2x* to emphasize the distinction; see Figure 3.9) and as such, is larger by better than an order of magnitude than the Earth's rotation rate. It is therefore sufficient (and more convenient)

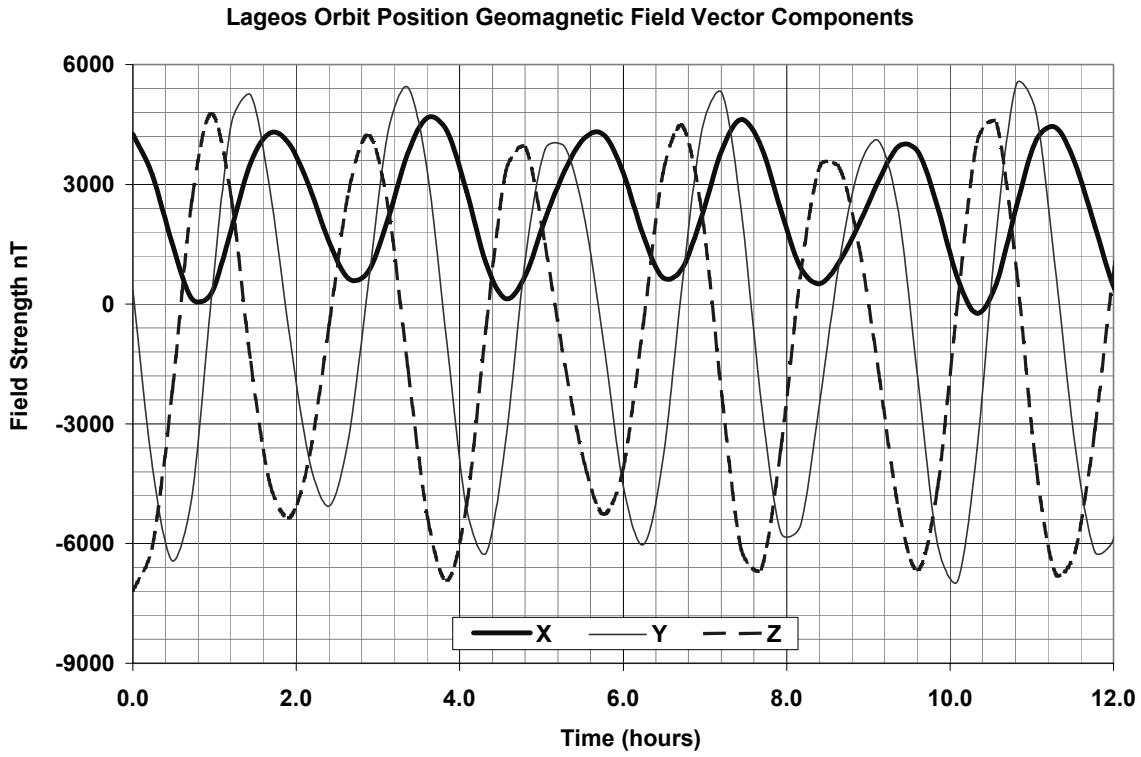

Figure 3.9 Sample geomagnetic field vector components at Lageos orbit positions

A representative sample of the geomagnetic field components experienced by Lageos throughout its orbit. The components are in phase with each other with a beat-frequency twice that of the orbital frequency. The data shown spans just over three orbit revolutions.

in the following discussion to ignore the effects of the Earth's rotation. If desired, the effect can be incorporated after the fact via the solution we propose.

For a conductor moving with velocity $\boldsymbol{v}$ through a time-dependent field, the generalization of the field equations (57) is

$$\nabla \cdot \boldsymbol{H}_i = 0$$

$$\frac{\partial \boldsymbol{H}_i}{\partial t} - \nabla \times (\boldsymbol{v} \times \boldsymbol{H}_i) = -\frac{i}{k^2 \omega} (\nabla \times (\nabla \times \boldsymbol{H}_i)) \qquad 79$$

where k is given by (54). L&L argue these reduce to the form of (57) when expressed in terms of coordinates in which the conductor is fixed. Unfortunately, however, the time-dependence of the external field is no longer a simple function of a single frequency nor can the variance of the field be considered constrained to two dimensions (the assertion leading to (52)). Therefore, the specific L&L results we reported previously no longer apply.

A rigorous treatment of this more general case does not lead to promising results, particularly in terms of numerical implementation. The system of equations for the multi-frequency case would themselves have to be treated numerically at a cost and complexity exceeding that of the larger model in which they reside. Therefore, we have chosen not to pursue this course for the current work.

Instead, we seek a compromise to bridge the gap between the existing solution and the true dynamics that are taking place. To proceed within the framework of the L&L solution, we explore two possibilities. The first treats the problem as if completely separable, computing independent solutions for the spin and the 2x orbital frequencies. The results are then linearly superposed to obtain the net magnetic torque on the satellite.

The second derives from the fact that the time dependence of (57) still can be resolved to a solution of the form $e^{-i\overline{\omega}t}$ for a possibly complex constant $\overline{\omega}$ via the method of separation of variables. To obtain $\overline{\omega}$ and the corresponding vector $\overline{\boldsymbol{\omega}}$, the independent angular velocity vectors are summed

$$\overline{\boldsymbol{\omega}} = \boldsymbol{\omega} + 2\boldsymbol{\omega}_{\mathrm{o}} \qquad 80$$

where $\boldsymbol{\omega}_o$ is the orbital angular velocity. Then $\overline{\omega}$ is simply the magnitude of $\overline{\boldsymbol{\omega}}$ and the L&L solution can proceed as before.

Both of these possible solutions take liberties with the nature of the physical system but we believe them to be reasonable in light of the abstractions already in place. For one, when the spin rate is much larger than the 2x orbit frequency, the latter contributes negligibly to the net torque under either approach. This preserves the desired behavior in the dominant frequency case. Also, for complimentary orientations of the spin and orbit angular velocities, both approaches provide conceptually accurate behavior. It therefore seems a logical extrapolation to expect reasonable results from these ideas. This is particularly true in comparison to the baseline approach, which completely ignores any effects of additional frequency components.

While either approach appears suitable to our needs, we have opted for the linear superposition solution to the multi-frequency problem as a selectable option in the W02 model. The implementation is identical to the case for the spin angular velocity substituting instead the 2x orbital angular velocity

$$2\boldsymbol{\omega}_o = 2\omega_o \begin{pmatrix} \sin i \sin\Omega \\ -\sin i \cos\Omega \\ \cos i \end{pmatrix}. \qquad 81$$

The value for ω_o is that of the net angular motion, η_v (see Table 3.2). The resulting torque expression is added to that for the spin frequency to generate a net magnetic torque for the model.

Summary

To address the deficiencies of the H&W model magnetic torque implementation, we evaluated the three primary ways the L&L reference sphere approach violates the physical system. To compensate for the geometric difference between the cylindrical core and its spherical approximation, a directional scaling of the coefficients of magnetization was introduced. The scaling magnitude is controlled by the parameter κ, and the directional scaling given as follows

$$\kappa' = \kappa \left(1 - \tfrac{1}{2} \left| \hat{\boldsymbol{B}} \cdot \boldsymbol{e}_3^{\mathrm{B}} \right| \right). \tag{67}$$

$$\begin{aligned} \widetilde{\alpha}' &= \kappa' \alpha' \\ \widetilde{\alpha}'' &= \kappa' \alpha'' \end{aligned} \tag{68}$$

The specific contribution to the net magnetic torque due to currents in the shell remains a mystery. However, we were able to show that for the most part, the effects of the shell can be accounted for in the parameterization above. Finally, to address the multiple frequency issue, the general problem is treated as separable. The torque contributions of the different frequencies are computed independently and linearly superposed to generate a net magnetic torque. Our implementation includes only the satellite spin and 2x orbit frequencies; the Earth's rotation could also be taken into account but it seems unnecessary to do so at this time.

3.6 Numerical Integration

With the physical model defined, we now address numerical implementation. This issue has already been a persistent consideration throughout the preceding discussion. For example, part of the criteria for pursuing particular solutions in the physical modeling effort has been the numerical feasibility of such approaches. Likewise, we have attempted to demonstrate efficient ways to construct algorithms based on the discoveries of the previous sections. Finally, the general issues of numerical application were highlighted in the discussion on artificial effects at the beginning of this chapter.

Now that the pieces have been assembled, however, it remains to "solve" the problem. That is, we must propagate by numerical integration the system described by (13)–(15) together with the torque expression we derived in the previous sections.

3.6.1 Requirements of the Numerical Problem

From a modeling standpoint, we are confronted with integrating an inhomogeneous nonlinear system of second order differential equations over an extended interval. While numerical methods exist for the direct integration of second order systems, these approaches have their greatest advantage when the first derivatives of the variables do not appear explicitly in the equations [Gill & Montenbruck, 2000]. The advantages of such approaches for the current situation where the equations of motion are populated with first derivatives is less certain. Accordingly, we follow the legacy of the H&W model and pursue a first order numerical integration solution.

Numerical solution methods for first order ordinary differential equations (ODEs) abound; the field is rich in history and well developed. The different families of ODE methods and different implementations within each family present distinct characteristics that may be more or less suitable for a given problem. Indeed, different solutions may be preferred for the same underlying system of equations depending on the goal of the particular task.

There are a number of concerns for the present problem but let us first recall the reduction of (13)–(15) to a first order system exhibited in Section 2.2.4. Namely,

$$\boldsymbol{Y} = (\theta \quad \phi \quad \psi \quad \dot{\theta} \quad \dot{\phi} \quad \dot{\psi}) . \tag{16}$$

and

$$\dot{\boldsymbol{Y}} = \begin{pmatrix} \dot{Y}_1 \\ \dot{Y}_2 \\ \dot{Y}_3 \\ \dot{Y}_4 \\ \dot{Y}_5 \\ \dot{Y}_6 \end{pmatrix} = \begin{pmatrix} Y_4 \\ Y_5 \\ Y_6 \\ \ddot{\theta}_{fr} + \ddot{\theta}_F \\ \ddot{\varphi}_{fr} + \ddot{\varphi}_F \\ \ddot{\psi}_{fr} + \ddot{\psi}_F \end{pmatrix} \tag{17}$$

where the free and forced motion equations are given respectively by

$$\begin{aligned} \ddot{\theta}_{fr} &= -\frac{Y_5 \sin Y_1}{I_1}\left[(I_3 - I_1)Y_5 \cos Y_1 + I_3 Y_6\right] \\ \ddot{\varphi}_{fr} &= \frac{Y_4}{I_1 \sin Y_1}\left[(I_3 - 2I_1)Y_5 \cos Y_1 + I_3 Y_6\right] \\ \ddot{\psi}_{fr} &= -\frac{Y_4}{I_1 \sin Y_1}\left[\left((I_3 - I_1)Y_5 \cos Y_1 + I_3 Y_6\right)\cos Y_1 - I_1 Y_5\right] \end{aligned} \tag{18}$$

and

$$\begin{aligned}
\ddot{\theta}_F &= \frac{1}{I_1}\left[N_1 \cos Y_3 - N_2 \sin Y_3\right] \\
\ddot{\varphi}_F &= \frac{1}{I_1 \sin Y_1}\left[N_1 \sin Y_3 + N_2 \cos Y_3\right] \qquad . \qquad 19 \\
\ddot{\psi}_F &= -\frac{\cos Y_1}{I_1 \sin Y_1}\left[N_1 \sin Y_3 + N_2 \cos Y_3\right] + \frac{N_3}{I_1}
\end{aligned}$$

For convenience, (17) is often written as $\dot{\boldsymbol{Y}} = \boldsymbol{f}(\boldsymbol{Y})$.

There are a number of general requirements for the numerical integration process that derive from the goals of the overall effort. There are also some issues and constraints imposed by the system of equations themselves. All of these ideas are reviewed in the sequel, beginning with the latter.

Direct Observations

First, while the previous sections make clear that the problem of resolving the torques is no simple issue, it can nevertheless be regarded as an external input to the numerical integration process. That is, N_1, N_2, and N_3 are seen by the integrator as numerical values rather than implicit expressions of the variables of integration. Thus, the preceding equations are a complete explicit statement of the problem in terms of the state variables.

On the other hand, the point should not be taken too far. For implicit integration methods, the torques must be accounted for in the computation of the Jacobian of the system. Because of the complexity of the torque equations, the Jacobian in this problem cannot be determined analytically so a numerical evaluation is required. The

corresponding additional computational burden makes implicit methods far less appealing.

The baggage associated with the Euler Angle approach should also be reiterated. Namely, the system is locally ill-conditioned for values of $\sin Y_1$ near the singularity at $\sin Y_1 = 0$. This starts to be a concern for Lageos when the angular velocity vector decouples from the body axis of the satellite and so is relevant for the not too distant future.

Nevertheless, we have not made any attempts in the present effort to explicitly deal with the issue. Instead, our focus has been an improved performance of the model versus historical observations when the singularity is not an issue. The concern can be addressed after the fact as a later enhancement once the dynamical issues are firmly resolved. This can be accomplished either by using two different inertial reference systems and switching between the two to avoid the singularity or by adopting a variable convention for which the singularity is not an issue (see Section 2.2.2). For the present discussion, it suffices to observe that individual numerical integration techniques may be more or less adept at handling singularities within the integration interval [Flannery et al, 1992]. This will be a factor in the evaluation of integration approaches for the problem.

Another factor influencing the choice of numerical integration method is the *stiffness* of the system. The formal definition of a *stiff* ODE is often of little practical use so informal characterizations abound (see, e.g., Ascher & Petzold [1998]). The particular

notion that is useful for the present system describes an ODE as stiff if there are multiple disparate timescales on which the state variables are changing.

For Lageos, this can be seen in terms of the spin and orbital frequencies. Early on, the large spin frequency dominates. This forces an inappropriate integrator to take excessively small steps in spite of the fact that the system is dynamically stable (slowly evolving spin state). The problem is stiff. Later, however, the timescales are more comparable—the system becomes decreasingly stiff due to the dissipation of energy. Since we are more concerned in the dynamics that occur later, it will be convenient to pursue non-stiff integration solutions.

As a final observation about the first order system, note that it makes essentially no demands on data storage because there are only six state variables. This makes methods for which data storage otherwise might be an issue, namely multi-step methods, feasible for the present system.

General Integration Requirements

Apart from the observations above based directly on the system of equations, it is also necessary to characterize the overall goals of the problem as they relate to the numerical integration. Typical concerns include accuracy, stability, efficiency, and practicality. There is also a type of physical constraint placed on the integrator that derives from the nature of the system. We take a closer look at each of these ideas:

- Accuracy. The spin state of Lageos will often be propagated over lengthy time periods. This puts a premium on global accuracy and therefore also local

accuracy. Thus, a method (or methods) flexible enough to handle tight error tolerance requirements is necessary, i.e., we require a high order method.

- Stability. On the other hand, there is also an implicit stability requirement when extended integration intervals are used. Unfortunately, there is often a direct trade-off between the order of the integration method and its stability [Gremaud, 2000]. By itself, the stability concern would also tend to push us toward implicit integration methods, but other considerations discourage their use. In the end, we favor accuracy as a higher criteria and so will lean more toward an a-posteriori determination of "reasonable" integrator behavior as a validation of stability.
- Efficiency. The speed of the method is always an overriding concern—all other things being equal, the faster the performance the better. Still, there are no "real-time" requirements placed on the present model so efficiency enters the consideration as an important but not primary issue.
- Practicality. It is not the goal of this effort to find (or invent) the *optimal* integration technique for our problem. Rather, we seek an approach appropriate to the task that can also be efficiently implemented into the existing software package. This leads us to seek more of a *black box* solution; i.e., mature non-proprietary integration package(s) that provide a simple interface. The practicality requirement tends to elevate explicit methods in general and one-step approaches in particular.

- Energy-Preservation. Given the nature of the system, it is appropriate to seek an integration solution that satisfies the general conservation laws governing the problem. In particular, in the absence of torques, the integrator ought to indefinitely preserve the total angular momentum of the system. This criterion provides a kind of litmus test for candidate integration techniques [Campbell, private communication].

With these requirements in mind, we now proceed to explore the types of solution methods available to find an appropriate match for our problem.

3.6.2 Survey of Integration Methods

In light of the preceding discussion, it should be clear that it is insufficient (and potentially dangerous) to simply employ the nearest available integration package without further consideration. Yet, to some extent, the H&W model took this route.[12] To be a little more deliberate about the process this time around, we first embark on a generalized overview of numerical integration methods.

Definitions

Already we have employed some terminology that suggests a categorization of approaches. *Multi-step methods* that compute the new solution based on a number of previous data points vs. *one-step methods* that use only the current system state. *Explicit*

[12] Other modeling efforts we have discussed are altogether silent on the issue so we are unable to ascertain the suitability of the method used.

methods where the new solution is an explicit function of the previous data vs. *implicit methods* that require a non-linear algebraic solve to isolate the new state. There is the *order* of the method that specifies the theoretical order of magnitude of the *local* error term (i.e. one step), and the *stability* of the method that, loosely speaking, is an evaluation of the complex domain over which a simplified test problem, $y' = \lambda y$, leads to convergent behavior within the method.

Other factors important to the general discussion of numerical integration methods include the *stiffness* of the problem (discussed above), the use and methodology of adaptive step size control, and the manner in which error and error tolerance is computed. Adaptive stepsize control is an efficiency feature—an optimization of a given approach. The goal is to modify the integration step size so that the integrator works just hard enough, but not too hard, to achieve the specified accuracy tolerances. In particular, it controls the current step size so that the internal estimation of error for each state variable is smaller than a computed error tolerance (below). And, it sets the next step size based on the relative difference between the estimated error and the corresponding error tolerance.

The internal estimation of error for a given integration step is specific to the integration method and is, in fact, another basis on which methods are sometimes categorized. On the other hand, the local error tolerance (ETOL) is derived from user specified absolute (ATOL) and relative (RTOL) error tolerances together with the current state and perhaps the current derivative estimate as well. The tolerances may be specified

per state variable or as global values. A generic form for the *j*th state variable is given by Booth et al [2001]

$$ETOL_j = \tau\left[ATOL_j + RTOL_j\left(a_y \mid Y_j \mid + a_{dydx} h \mid f_j(\boldsymbol{Y}) \mid\right)\right] . \qquad 82$$

where $\tau \leq 1$ is a scaling "fudge factor", h is the integration step size, and a_y and a_{dydx} are weight factors. All the parameters are considered non-negative. Though it represents a specific customization of (82) (roughly, $\tau = a_y = a_{dydx} = 1$, $\text{ATOL}_j = 0$), the form suggested by Bulirsch & Stoer [1993] is perhaps more intuitive

$$ETOL_j = RTOL_j \cdot \max\{\mid Y_j(t) \mid : x \in [t_o, t_o + h]\} . \qquad 83$$

This is a relative scaling of the largest possible value of the derivative function within the integration step interval.

The full form of (82) is rarely employed. Usually (depending on the integration package), some of the parameters are hard-wired and some are available as integrator "control knobs." A common choice, and the one that shows up most frequently in the packages we considered, is the form

$$ETOL_j = ATOL_j + RTOL_j \mid Y_j \mid . \qquad 84$$

We have further simplified the implementation by specifying only one independent absolute and relative tolerance. That is, $\text{ATOL}_j = \text{ATOL}$ and $\text{RTOL}_j = \text{RTOL}$ for all j.

Following the form of Gill & Montenbruck [2000], we outline three basic families of methods: i) *Runge-Kutta methods*, ii) *Extrapolation methods,* and iii) *Multi-step methods*. Based on the earlier discussion, we restrict ourselves only to the explicit forms of these methods.

Runge-Kutta Methods

Runge-Kutta or RK methods might be described as the workhorses of numerical integration. They are generally efficient, but are not the fastest. They may be broadly applied with considerable confidence. RK methods are more tolerant of difficult phenomenon such as singularities or discontinuities in the derivative equation $\boldsymbol{f}$. And, they are fairly easy to implement. As a result, RK methods are often the first to be applied to a given ODE..

The general approach is simple enough; the inspiration derives from the first order Taylor expansion of $\boldsymbol{f}(\boldsymbol{Y})$

$$\boldsymbol{Y}(t_o + h) = \boldsymbol{Y}_o + h\, f(t_o, \boldsymbol{Y}_o) + \ldots \tag{85}$$

leading to *Euler's Method*

$$\boldsymbol{Y}_{n+1} = \boldsymbol{Y}_n + h\, f(t_n, \boldsymbol{Y}_n) \equiv \boldsymbol{Y}_n + h\, f_n + \mathrm{O}(h^2) \tag{86}$$

which is a first order method (error term one order higher than the correction term). The idea extends by considering Euler type steps not across the entire interval h as in (86), but rather as sub steps within the interval. For example, the *explicit midpoint method* is obtained by using an Euler step (86) to estimate the state at the midpoint of the interval, then using the derivative at the point to take a step across the entire interval

$$\begin{aligned} k_1 &= \boldsymbol{Y}_n + \tfrac{1}{2} h\, f_n \\ k_2 &= \boldsymbol{Y}_n + \tfrac{1}{2} h\, f(t_n + \tfrac{1}{2} h, k_1)\,. \\ \boldsymbol{Y}_{n+1} &= \boldsymbol{Y}_n + k_2 + \mathrm{O}(h^3) \end{aligned} \tag{87}$$

Note that the result is a second order method. This generalizes further (see e.g., Ascher & Petzold [1998]) so that $\boldsymbol{Y}_{n+1}$ is expressed as a linear combination of any number (but fixed for a given approach) of intermediate evaluations. The method is called an *s*-stage Runge-Kutta method if *s* such intermediate evaluations are used and it turns out that the order of the method, *p*, cannot be greater than *s*.

Adaptive step size control is added by combining two RK methods of subsequent order, *p* and *p*+1, and using the difference of the computed solutions as an estimate of local error. In special (ideal) circumstances, the two methods share the same intermediate computations, and so save costly evaluations of the derivative function $\boldsymbol{f}$. Such approaches are called *embedded* methods and are the basis for most practical implementations of explicit RK approaches.

Once a particular RK method is employed, the order is fixed throughout the application. This locks in a range of suitable relative accuracies (i.e., tolerances for which the method is relatively efficient). Gill & Montenbruck [2000] assert that for high accuracy work (≥ ~8 digits) an RK method of order 8 or higher is generally necessary. This conclusion is specific to the types of problems they investigate but the analysis used satellite orbit equations and so has a strong correlation to our own system.

Extrapolation Methods

Extrapolation methods, also called "Bulirsch-Stoer methods" due to the pioneering work of Bulirsch & Stoer [1966], are based on the notion of Richardson extrapolation. The idea is to traverse the interval *h* with a suitable low-order method using increasingly fine

substeps, $h_i = h/m$. The resulting estimates at $t+h$ are treated as a function of the substep size and an extrapolation is performed to predict the limiting result for zero step size.

The appeal of the approach can only truly be seen in the details for which we refer to Gill & Montenbruck [2000] or Flannery et al [1992]. We summarize some of the more informative results. Starting from $(t_n, \mathbf{Y}_n)$, the system state is advanced to $t+h$. The interval $[t, t+h]$ is divided into an increasing number of substeps of size $h_i = h/m_i$ defined by the *Bulirsch sequence*

$$m = m_1, m_2, m_3, \ldots . \tag{88}$$

The interval is traversed using a *modified midpoint rule* consisting of an Euler step (86) followed by midpoint method steps (87)

$$\begin{aligned} z_1 &= \mathbf{Y}_n + h_i f(t_n, \mathbf{Y}_n) \\ z_{j+1} &= z_{j-1} + 2h_i f(t_n + jh_i, z_j) \qquad (j = 1, \ldots, m_i - 1) \end{aligned} \tag{89}$$

and the approximate solution at $t+h$ is given by

$$\mathbf{Y}_{n+1}^i = \tfrac{1}{4} z_{m_i - 2} + \tfrac{1}{2} z_{m_i - 1} + \tfrac{1}{4} z_{m_i} . \tag{90}$$

The fundamental discovery at the core of the extrapolation method is that the error in (90) is an *even* power series of the substep size, h_i, with coefficients that depend only on t_n and h but not on h_i. This means that approximations of the form (90) corresponding to subsequent members of the Bulirsch sequence can be linearly combined to eliminate the leading error term in the power series. This results in a refined solution estimate that is better by two orders of magnitude. The idea generalizes—any two such subsequent refined approximations may themselves be linearly combined to eliminate the next

leading term, and so on. The result is a triangular sequence whose (i, i) entry is an approximate solution to the ODE at $t+h$ with error $O(h^{2i+1})$.

Just as with the RK methods, local error is estimated by differencing two subsequent solution approximations. Likewise, adaptive stepsize techniques use this error information to monitor the performance. However, the stepsize control typically also accounts for how 'deep' into the Bulirsch sequence the process went before reaching an acceptable value. The implementation of the stepsize control and the specific values in the Bulirsch sequence (other than that they must all be even) are the two primary factors that distinguish extrapolation techniques.

Extrapolation methods are variable order as can be seen from the error term. This is a powerful advantage when accuracy is a premium. For arbitrary precision arithmetic, extrapolation methods can attain accuracies well beyond any known RK implementation. Likewise, the variable order extrapolation approaches may be better suited for problems in which a wide range of tolerances are likely to be specified; although, there is a diminishing return to efficiency gained when relaxing tolerances to the low end of the spectrum. Moreover, for a given accuracy requirement (or range of requirements) for which a suitable RK approach exists, the RK method will almost always be more efficient than any extrapolation technique. Finally, extrapolation methods have a much more difficult time with irregularities in the derivatives.

Multi-Step Methods

In the previous examples, the solution at each integration step t_n+h used only the information from the problem at time t_n. There is an implicit flexibility to this approach and it has obvious advantages when data storage is an issue. Nevertheless, significant efficiencies can be gained for a given accuracy requirement if past values are employed in the computation of subsequent integration steps. This is the motivation behind multi-stop methods and it provides the basis for a host of possible implementations. As in the previous sections, it is not our goal to rigorously develop the theory of multi-step methods but rather to simply highlight some of the main points of interest.

A general linear k-step multi-step method for the numerical integration solution at time t_n can be expressed in the form (see e.g., Ascher & Petzold [1998])

$$\sum_{j=0}^{k} \alpha_j \boldsymbol{Y}_{n-j} = h \sum_{j=0}^{k} \beta_j \boldsymbol{f}_{n-j} \,. \tag{91}$$

where α_j and β_j are coefficients specified by the particular method and h is a *fixed* step size. It is customary to set $\alpha_o = 1$ and we note the method is explicit if $\beta_o = 0$ and implicit otherwise.

Particular groupings of coefficient choices separate multi-step methods into families. The most popular multi-step family (almost to exclusion) for non-stiff problems are called Adams methods which have the form

$$\boldsymbol{Y}_n = \boldsymbol{Y}_{n-1} + h \sum_{j=0}^{k} \beta_j \boldsymbol{f}_{n-j} \,. \tag{92}$$

The explicit forms of (92) are called Adams-Bashforth (A-B) methods while the implicit versions go by Adams-Moulton (A-M); with k steps, these have orders k and k+1 respectively. Unfortunately, the stability properties of the A-B methods are not very nice, particularly as the order increases. The A-M methods, on the other hand, have much larger stability regions but suffer the aforementioned disadvantages of implicit techniques.

A well-known and widely applied compromise is a group of methods identified as *predictor-corrector*. The explicit A-B method is used to provide a "low quality" prediction ("P") for $\boldsymbol{Y}_n$. The prediction is then used to estimate ("E") $\boldsymbol{f}_n$ so that the A-M method can provide a correction ("C") for $\boldsymbol{Y}_n$. The correction feeds back into a revised estimate ("E") for $\boldsymbol{f}_n$. This so-called ABM PECE method is a k+1 order method with stability properties somewhere between A-B and A-M [Gremaud, 2000].

In problems where a fixed stepsize is suitable, this approach is typically far superior from a computational standpoint to that of either the RK methods or extrapolation techniques. But what of problems where adaptive step control is an implicit requirement of the underlying system (as is the case for our problem)? Fortunately, generalizations exist to the PECE method that allow not only adaptive step control but also employ variable order. This is an obviously desirable combination that makes such approaches appealing candidates for our present application.

Summary

Though this survey of numerical integration issues and techniques is far from exhaustive, we have tried to highlight some of the important and useful features of the topic. Each of the general families of methods reviewed have certain appealing properties; enough so that none can be considered unilaterally superior. To summarize (see Bulirsch & Stoer [1993]):

- RK methods are easy to employ and widely adaptable. They achieve modest accuracy but are perhaps best able to handle discontinuities and singularities in the equations.
- Extrapolation methods can provide extremely accurate results but also often provide more accuracy than required at the expense of efficiency. They are particularly ill-suited for singularities.
- Multi-step methods in general and the variable-order variable-step PECE approach in particular are extremely efficient. However, more operational overhead per step is required so the advantage is lost when the RHS of the differential equation is inexpensive to compute.

Each of these ideas has advantages and potential drawbacks for the present problem. We therefore have implemented a solution from each family into the W02 model. More details are provided shortly but first we examine the method originally deployed by the H&W model.

3.6.3 H&W Model Integration Method

The H&W model employs an extrapolation technique adapted from [ii]. The source uses an adaptive step technique attributed to Deuflhard [1983] with a Bulirsch sequence

$$m = 2, 4, 6, 8, 10, 12, 14, \ldots. \qquad 93$$

The details of the adaptive step control method are rather involved and so we refer to Flannery et al [1992] for the details.

The modified version introduced in the H&W model uses the original Bulirsch sequence of Bulirsch & Stoer [1966]

$$m = 2, 4, 6, 8, 12, 16, 24, 32, 48, \ldots. \qquad 94$$

and a simplified adaptive stepsize control based only on the index of the Bulirsch sequence; it does not directly use the local error estimation. In particular, the method sets a "goal" of achieving an acceptable integration step solution at the 5^{th} or 6^{th} element in the Bulirsch sequence. If the result instead occurs at the kth element, the subsequent step is adjusted by the factor m_6/m_k. The approach is perhaps more intuitive than the Deuflhard algorithm in Flannery et al but is also theoretically less efficient.

3.6.4 W02 Model Revisions

The advantages of the preceding approach have already been highlighted—a high accuracy method that is suitable for long integration times. Moreover, we can verify based on the correlation of the output to empirical data that the results are reasonable. Finally, we tested the energy preservation properties by integrating the free-motion

equations for ~75 years and observed no change in angular momentum. Thus, the H&W extrapolation method is an acceptable, if not ideal, numerical integration solution for the Lageos equations of motion.

Still, the approach has some weaknesses. Of particular concern is the noted difficulty with singularities—a more robust solution is preferred. In addition, because extrapolation methods tend to be somewhat inefficient, it is reasonable to seek a faster solution.

To address these concerns, we pursued a number of ideas. First, we added the Deuflhard extrapolation algorithm to the model to investigate whether, by comparison, the simplified adaptation of the H&W model was a significant source of efficiency loss. Next, we initiated a search for suitable integration packages to improve upon the H&W model's limited extrapolation implementation. Motivated by the ideas presented above, we opted to include both a high order RK method and a variable order variable step ABM PECE method. To reduce the field of candidates in these families (see, e.g., [T]), we followed the lead of Gill & Montenbruck [2000] who evaluate the performance of a host of integration techniques for applications in orbit mechanics. Their results lead us to an appealing choice in each category that also match our practicality concerns. In the end, we added three integration modules as selectable options within the W02 model:

- *Deuflhard extrapolation method* (BD) from [ii];
- 8th order embedded *Dormand & Prince Runge-Kutta method* (DOPRI8) from [iii];
- Variable-order, variable-step *Shampine ABM PECE method* (DE) from [iv];

We also retained the existing method

- *Bulirsch-Stoer extrapolation method* (BS) adapted from [ii] and Bulirsch & Stoer [1966].

DOPRI8 is credited to Hairer (see Hairer et al [1993]) and claims to be best suited for 7 to 13 digit accuracy requirements. Our implementation uses a C version of the routine translated from the original Fortran. The interface is straightforward and the package easily integrated into the existing C structure of the W02 model. DE is due to Shampine (see Gordon & Shampine [1975]). The code is a Fortran subroutine that required some additional work to incorporate into the model (see Section 3.7) but otherwise provides a clean calling structure. The infrastructure for BD already existed in the model making its implementation straightforward.

We tested each of the new methods using a ~75 year interval to ensure they met the energy preservation litmus test (they did). We then analyzed and compared all the methods' outputs over both short and relatively long term data sets and with multiple specified tolerances as a sanity check on the quality of the data. Spin state projections were comparable in all cases so we feel confident that each of the approaches is suitable for the present work.

We also evaluated the performance of each method for the system. The results are illustrated in Figure 3.10 and summarized below.

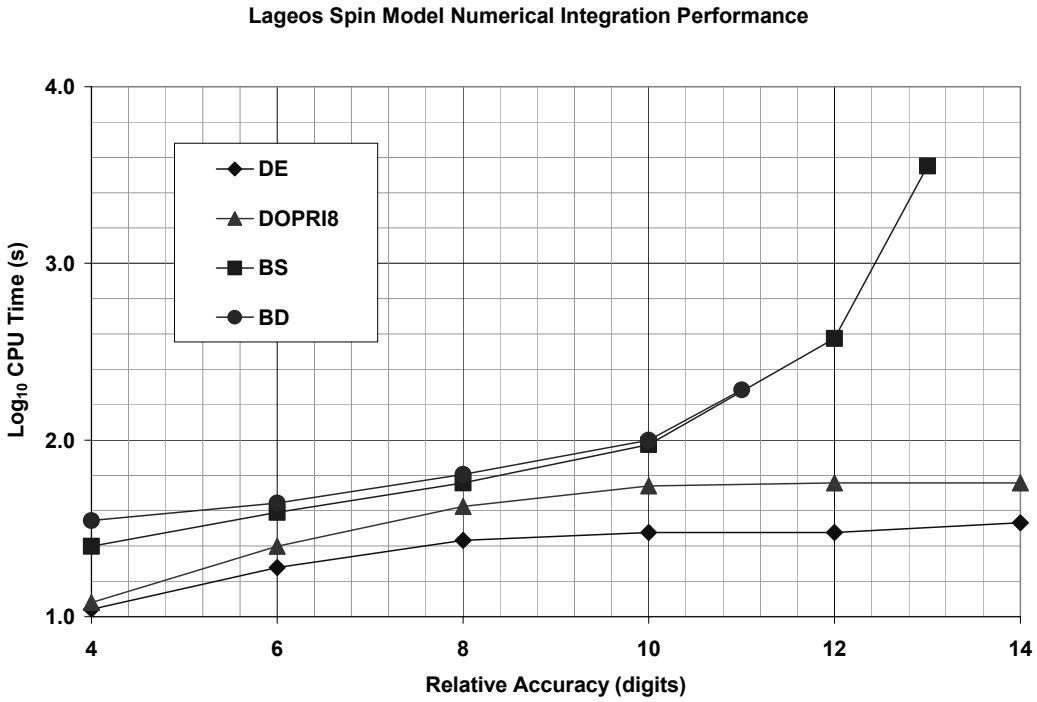

Figure 3.10 Performance comparison of the W02 model numerical integration packages

The model run time (Log base 10) is plotted against relative accuracy (RTOL) for the four numerical integration packages available in the W02 model. The test case used a 50 JD integration interval and included the IGRF octupole magnetic field, the J2 gravity gradient correction, and the multi-frequency magnetic torque correction. BD and BS failed to complete the integration at 12 and 14 digit accuracies respectively due to a step size underflow.

- The BD method did not show a particular performance advantage versus the BS approach. In fact, the results of several test cases are mixed at best and may even favor the BS method over all.
- The extrapolation techniques did not fare well for the higher relative tolerances. This would seem to conflict with the earlier assertions about the approaches but in fact, it might have been expected. Because these methods can provide arbitrarily

precise results, they also suffer more severely when the underlying system cannot provide sufficiently accurate information. Indirectly, then, the results show the precision limit of the model itself is $\sim 10^{-12}$ and so it is pointless to specify a tighter tolerance.[13]

- As expected, the DOPRI8 routine exhibits solid performance that betters the extrapolation methods for comparable tolerances but falls short of the multi-step approach.
- The results suggest DE and DOPRI8 squeeze out the utility of the extrapolation approaches. For pure performance, DE is preferred, as long as the derivative function is 'nice'. To handle the irregularities in the derivative function (singularities), DOPRI8 is the best choice. The extrapolation methods are left out in the middle.

3.7 General Software Enhancements & Features

So far, we have presented the theoretical foundation for the physical model and even outlined algorithms for numerical implementation. Likewise, approaches to numerical integration have been explored and specific methods identified for the model. There remain a host of issues and concerns related to the process of taking these building blocks

[13] These results suggest DE and DOPRI8 merely "pretend" to provide more precise results but are, in fact, unaware they have exceeded the precision of the system. In short, the output accuracy cannot be better than the input accuracy.

and producing a quality software model. In the preceding, we have emphasized our primary goal of providing a revamped model that is demonstrably improved in its predictive accuracy. A corresponding goal, however, is to generate a product that is usable for further study and accessible for the integration of future refinements. To this end, a number of topics related to the development and use of the model itself are now explored. Some of what follows is low-level detail but it is nevertheless important to document to ensure the integrity of the model's legacy.

3.7.1 Software Development Environment

The W02 model is written in the C programming language. This is largely due to legacy—the original model of Habib et al was written in C and all the subsequent revisions have followed suit. One of our goals, however, has been to maximize the use of previously developed and publicly available software to address the needs of the model rather than "reinventing the wheel." Unfortunately, many appealing tools are only offered in Fortran, which is the predominate language of numerical computing. In addition, some of the tools available in C are automated translations of original Fortran source that provide the functionality but not the efficiency of the parent

implementation.[14] This can render as ineffective an otherwise attractive numerical routine.

To combat this, we first searched for numerical tools written in native C (i.e., optimized for C). Unfortunately, this proved too restrictive; some appealing options were not available in C. We therefore took a different approach to the problem that has broader implications. In particular, the W02 Lageos Spin model is a multi-language implementation, combining both C and Fortran subroutines as necessary. The following packages are included in this work:

- GNU Scientific Library (GSL) [v] for general numerical architecture and specific packages (discussed in the sequel); C source.
- IGRF Geomagnetic Field model from the GEOPACK library [i]; Fortran source.
- DE numerical integration package [iv]; Fortran source.
- DOPRI8 numerical integration package [iii]; C source.[15]
- Numerical Recipes in C [ii] for numerical integration (BD and BS) and numerical differentiation routines; C source.

---

[14] The underlying algorithms for numerical routines are language neutral. However, optimal implementations will necessarily differ from one programming language to another as language specific architectural constructs are taken into account. Thus, an automated (dumb) C translation of a numerical routine optimized for Fortran will not be an optimal C implementation. Better to translate the algorithm "by hand" to ensure an optimal language specific routine.

[15] It was previously mentioned that the DOPRI8 routine we employ is a Fortran to C translation. In this particular case, the translation was done "by hand" and so it claims to be an efficient C implementation.

The main handicap of the multi-language approach is that it makes the task of writing a fully *portable*[16] model much more difficult. Moreover, even some of the C language tools present with *platform dependent* implementations that are much more easily utilized than their language independent versions. Because we are not software engineers, these factors conspired to make the pursuit of full portability time-prohibitive.

Instead, we employ a "pseudo" portability by utilizing a software development environment that is a) freely available, b) based on a compiler package that is widely circulated, and c) natively supports multi-language programming. We proceed hoping that the accessibility of these development tools will allow future work on the model to progress without difficulty. However, we acknowledge that those committed to a specific platform different from ours may find considerable work must be done to adapt the W02 model to their needs.

Dev-C++ Integrated Development Environment

Our platform is a PC compatible system running the Windows 2000 operating system. We use an open source (free) integrated programming development environment (IDE) titled Dev-C++ [vi] from Bloodshed Software.[17] Dev-C++ is an excellent programming tool for 32 bit Windows (Win32) operating systems packed with useful features to

[16] That is, code that is transportable without modification to any *platform* (computer hardware, operating system, and programming development tools such as the compiler, linker, and etc.) that satisfies only very general conforming requirements.

[17] The French author apparently just likes the phonetic sound of "Bloodshed" and appreciates the connotation of the "blood, sweat, and tears" involved with software development.

facilitate quality software development. Two things about this package are important for the present discussion: i) Dev-C++ utilizes a compiler suite—the GNU Compiler Collection (GCC)—that is widely used and natively supports multi-language programming [AA], and ii) Dev-C++ includes excellent *project management* capabilities making it particularly easy to integrate stand-alone modules (including those in languages other than C).

GCC – GNU Compiler Collection

To be specific, Dev-C++ uses the MinGW (Minimalist GNU for Windows) port of GCC [Z]. MinGW is a free (to use and distribute) collection of GNU programming tools [DD] adapted to the Win32 environment. Software developed on Win32 platforms using these tools is fully portable within the GNU family. It is therefore appropriate to peel off the pre-packaging (Dev-C++ & MinGW) and focus directly on the features of GCC.

As the name from which the acronym derives implies, GCC is actually a collection of compilers integrated into a single package. More precisely, GCC is a collection of language specific front ends that use the GCC compiler. To be sure, the front ends are not merely translators but invoke language appropriate compilation from GCC. The supported languages include C, C++, Fortran, Java, and Ada. Each can be invoked by directly calling "gcc" (this is transparent in the Dev-C++ IDE) but when a particular emphasis is desired, the specific front ends are accessed respectively as "gcc" for C, "g++" for C++, "g77" for Fortran, gjc for Java, and gnat for Ada.

The support for each included language is full featured and conforming. That is, GCC provides all of the standard libraries and toolsets and fully supports the language standards. In addition, GCC provides a number of practical language extensions, though use of these will of course make code less portable. Further details regarding standards and extensions are beyond the scope here (except to note the conventions used) so we refer to the GCC Reference Manual section on Standards [CC] (also see [EE]).

C99 Language Features

The W02 model makes use of the multiple language facilities (detailed below) and we have adopted a number of features in the C99 standard [GG] that are not part of the current ANSI C construct. Specifically in terms of the latter, we make extensive use of the following C99 features:

- Variable length arrays (VLAs) in lieu of the more cumbersome malloc procedures of ANSI C;
- Inline capability for faster code;
- Mixed declarations to localize the scope of automatic variables; and
- The C++ style "//" comment indicators for convenience.

These capabilities are made possible by invoking a specific compiler flag. In our version (GCC 2.95.3–6), the flag is "-std=gnu9x." There are several more recent versions of GCC that use the flag "-std=c99." It was not until well into our analysis, however, that

any of these newer versions of GCC were ported to Win32 by MinGW so they were not part of the IDE we used.

Multi-Language Integration

A major benefit of the GCC package is the ability to mix raw source code from different languages in the same project. It remains to specify an interface that allows (in our case) C and Fortran subroutines to communicate with each other because Fortran and C subroutines use different calling conventions and data structures. Fortunately, for each of the Fortran source-code packages we use, there is a single point of entry so the difficulty is minimized.

This issue of mixing Fortran and C in particular has received quite a bit of attention (see e.g., [HH], [II], and Burley [2002]) and the main difficulty when calling Fortran subroutines from C is prototyping the calling structure (subroutine names and data types). The embedded interface between Fortran and C within GCC is based on the automated Fortran to C translator/compiler "f2c". We have already noted our bias against automated translations of Fortran code to C, which is why we did not directly pursue f2c as a solution to the mixed language problem. Here, however, the f2c constructs merely provide the necessary interfaces to allow the Fortran and C subroutines to "speak" to each other (recall GCC performs native compilation for each language).

To construct the appropriate C prototypes for the Fortran subroutines we followed the idea suggest in the GCC Fortran manual ([KK], see also [BB] or Burley [2002]). First, C prototypes (including data declarations) of the Fortran source subroutines can be

generated by using the command line f2c with the -P switch (e.g., "f2c source.f -P"). In the calling C routine, it is helpful to mimic the Fortran data types in the variable declarations to ensure compatibility. In addition, two libraries, libg2h and libm, containing the language interfaces must be included via linker flags "-lg2h -lm" (must be in that order).

Once these steps are accomplished, the Fortran and C source code may be included together in the same project. The prototypes for the Fortran subroutines determined above are otherwise handled in the same manner as prototypes for C subroutines.

3.7.2 GNU Scientific Library

One of the most significant architectural improvements of the W02 model has been the incorporation of the GNU Scientific Library or GSL [v]. GSL is an extensive free collection of ANSI C routines for numerical computing. In addition, GSL provides a number of constructs that make implementation of mathematical procedures more accessible. Once again, we have used a Win32 port of GSL as provided by the GnuWin32 project [W]. It just so happens (not quite by accident) that the GnuWin32 project performs all of its developments using the MinGW port of GCC. This makes GSL a particularly good fit for our work and certainly assures that our "pseudo portability" goal is met.

The specific GSL related enhancements we made in the W02 model include

- A complete scrub of data structures to incorporate the GSL Vector and Matrix constructs.

- Extensive utilization of GSL's Basic Linear Algebra Subprograms (BLAS) for vector and matrix operations resulting in a cleaner mathematical implementation compared to the cumbersome approach of the H&W model.
- A multi-dimensional non-linear minimization capability was added to allow parameter optimization (see below).
- Accompanying the optimization feature is a linear least squares fitting routine and a numerical differentiation routine.
- Incorporation of various efficient mathematical operations including integer power computations, number testing macros (maximum, minimum, sign), Horner's stable polynomial evaluation method, and a vector element sort routine.

As a result of these adaptations, the W02 code is not only cleaner and easier to decipher but also more capable and efficient.

The addition of the GSL package to the model is relatively straightforward. The GSL libraries, libgsl and libgslcblas, are attached to the model by inserting the linker commands "-lgsl -lgslcblas." Additional significant efficiencies are gained by instructing "inline" compilation of the GSL routines. This is accomplished by inserting the macro "-DHAVE_INLINE" as a compiler flag. Specific numerical routines are accessed by attaching the appropriate header file as specified in user's manual (Booth et al [2001]). For example, basic math operations and constants are added by including "gsl_math.h" wherever the functionality is utilized.

As a final remark about GSL, we note that it contains a host of numerical routines that may be useful to future work with the Lageos Spin Model. Since the library is already built into the W02 model, it is a trivial matter to access additional routines as the need arises.

3.7.3 Parameter Optimization

One of the persistent themes in the preceding sections covering the dynamical features of the model is the idea that there is an element of tuning in selecting the values describing physical characteristics of the satellite. For example, the effective conductivity and radius of the reference sphere in the magnetic torque model were chosen so the model output would match the observed exponential decay of the spin rate. However, in the H&W model, these values were obtained merely by trial and error. Moreover, we feel the notion of tuning the parameters was not applied broadly enough. In response to these "deficiencies," we have added to the W02 model the option to perform a non-linear optimization on the satellite's model parameters.

Apart from the dynamical improvements summarized in earlier sections, we think this addition represents the most important advance of the W02 model. For one, it improves the ability to ask "what if?" questions and makes it possible to better understand potential sources of error. More importantly, it has the potential to significantly improve the predictive performance of the model by taking some of the conjecture out of the selection of model parameters.

The parameter optimization is implemented as a selectable option in the model using GSL's Multidimensional Minimization subroutines. Error is computed by comparing model output to the empirically determined Avizonis data and the procedure seeks parameter values that minimize this error. The optimization can be run over any duration of time interval as long as it spans some portion of the Avizonis data set.

We programmed six model parameters into the optimization (though more are easily added) and the software allows the choice of any combination of these parameters for a given optimization run. The six parameters are the principal moments, I_1 and I_3, and four values from the satellite core reference sphere: i) radius, a, ii) effective conductivity, σ, iii) magnetization coefficient scaling factor, κ, and iv) an oblate spheroid factor we experimented with early on but now believe can be deprecated.[18]

The Avizonis data identifies spin axis right ascension, declination, and spin rate at specific epochs. We compute spatial error for the optimization routine by differencing model outputs for right ascension and declination from the Avizonis values and summing their squares. The error associated with the spin rate is determined by computing the decay coefficient for the model output and comparing it to the empirically determined

[18] The oblate spheroid parameter derived from the idea that the cylindrical core is more similar to an oblate spheroid than a pure sphere. We defined an oblateness parameter as the ratio of the equatorial and polar radii of the spheroid and used this to perform a simple transformation mapping the spheroid to a pure sphere (z scaling). We applied the transformation to the body frame $\boldsymbol{B}$ and $\boldsymbol{\omega}$ vectors before proceeding with the L&L computations; the inverse transformation was applied to the resulting torque. This produced some nice results, but is redundant with the directional scaling of the coefficients of magnetization presented earlier. We think the latter is more elegant so the former is deprecated and is not included in any results.

decay constant. In particular, we take the log of the model output spin rates and perform a linear least squares fit to determine the slope (which is the decay coefficient). This leads to three error factors: the sum of squares of the right ascension error, the sum of squares of the declination error, and the square of the decay rate error. The three error terms are weighted per user input and summed to produce a total error.

Table 3.5 Satellite parameters featured in the optimization routine

I_1	Transverse principal moment of inertia
I_3	Axial principal moment of inertia
a	Radius of the core reference sphere
σ	Effective conductivity of the core reference sphere
κ	Magnetization coefficient scaling factor
f'	Core oblateness factor (deprecated)

The GSL Multidimensional Minimization subroutines feature three different conjugate gradient type algorithms and a steepest descent method (the latter is not efficient and only included for demonstration purposes). Two of the three conjugate gradient algorithms use a line-minimization technique, while the third is a quasi-Newton method. We did not perform a comparative analysis on these methods so cannot say which is best for our problem, but suspect the quasi-Newton method may not be appropriate because it puts too much confidence in the quality of the computed gradient (see below).

Each of the minimization routines require a gradient computation. Given the nature of the problem, there is no analytical solution so a numerical differentiation technique is required instead. Unfortunately, GSL is somewhat limited in this area—the derivative

routines do not allow enough control to make them useful for our current problem. The routine provided in [ii] has better practical utility and that is now the default approach in the optimization package. Two important observations are made about the numerical differentiation in this problem:

- First, the computation is extremely expensive and, unfortunately, the cost is unavoidable. Numerical derivatives are computed with some form of finite differencing. This requires function evaluations at several points bracketing the point of interest. For the parameter optimization, a single "function evaluation" consists of propagating the spin state over a specified interval, then computing the total error of the output as previously described. Depending on the duration of the interval and on the modeling options selected, each of these function evaluations can take from tens of seconds to several minutes (or more). At least six (usually more) such evaluations are required for a decent derivative estimate for a single parameter. Multiply by the number of parameters and perform the operations several times per optimization step and the cost is staggering. We have had optimization runs last from several hours to several days on our system.
- Second, the computed derivative cannot be expected to have much precision. The best-case theoretical accuracy of numerical differentiation is about half to two-thirds as many digits as the relative accuracy the function can provide (see Flannery et al). For our problem, the relative accuracy of a given data point output from the model is ~*nstp*·RTOL where *nstp* is the number of integration

steps. For DE, *nstp* scales to $O(10^4)$ integration steps per Julian Day of simulation time. If ten digit accuracy (RTOL = 10^{-10}) is specified and given a typical integration interval spans tens if not hundreds of Julian Days, the error function will have roughly four quality digits which means the derivative will be accurate to two or possibly three digits.

The implication of the second point is a weakening of the convergence properties of the minimization methods because each uses the magnitude of the gradient as an indicator of progress. The situation is particularly troubling for the quasi-Newton method, however, because it uses second derivative information derived from the gradient to attempt Newton-like steps toward the minimum.

The parameter optimization capability is implemented as a stand-alone feature of the model with its own code and control files. This is done to keep a clean user interface to the core feature of the model—the spin state propagation. Results and analysis of the parameter optimization efforts are discussed in Chapter 4.

3.7.4 Miscellaneous Features & Enhancements

In addition to the more significant modifications mentioned above, we also made a number of other enhancements that improve the overall quality of the W02 model. These are briefly summarized for completeness.

Time Format

Prior to the W02 model, integration was performed using a "local" time format with $t = 0$ corresponding to the initial conditions. Unfortunately, in this format there is a unique set of model parameters initial conditions corresponding to different epochs. This made switching to different sets of initial conditions cumbersome. This deficiency was corrected by adopting a global time format using the JD2K timescale. Integration start and stop times are specified according to their JD2K value and the initial spin state (Euler angles and rates) is set accordingly.

Targeted Output Times

Another limitation of the H&W model was an inability to specify times for data output. Instead, data was generated only at a fixed interval throughout the integration. Not only did this lead to copious unnecessary output, but it also made it difficult to correlate model outputs with empirical observations in both space and time. The latter point is particularly important because it renders as useless the parameter optimization routine. It was therefore necessary to add a targeted output time feature to the W02 spin model. Any set of times (JD2K) may be specified, but the default list in the model corresponds to the Avizonis data set. The times are automatically filtered to include only those within the integration interval.

Data Files

Along the same lines, the H&W model had very limited data output capabilities. Moreover, the format of the data files was inconvenient and required a great deal of post-processing. We modified this behavior to produce a much more usable set of output files. All of the output data is time tagged (JD2K). A sample of each output file is included in the appendix; the contents are are follows:

L_euler.txt: Euler angles (deg) and Euler angle rates (deg/s). ϕ and ψ are modulated to a user specified interval and the corresponding "revolution" numbers are provided. Also reported is the right ascension (deg) and declination (deg) of the body axis.

L_angvel.txt: Magnitude of the angular velocity ("spin") vector (deg/s) along with its body frame axial and transverse components (deg/s). Also provided is the spin vector's longitudinal and latitudinal angles (deg) in the body frame, the ECI frame (i.e., right ascension and declination), and an orbital reference system defined by the Euler angle transformation $\phi = \Omega$, $\theta = i$, and $\psi = 0$.[19]

L_angmom.txt: Spin angular momentum vector data in precisely the same output formats as L_angvel.txt.

---

[19] This frame is nearly depicted in Figure 2.4. The *x*-axis is along the line of nodes and the *z*-axis is the direction of the orbit angular momentum vector. The orbital motion occurs in the *x*-*y* plane of this system.

L_orbit.txt: Instantaneous orbit position data including radius (km), RAAN (deg), modulated net angular position (deg) and corresponding revolution number, mean anomaly (deg), eccentric anomaly (deg), and argument of perigee (deg).

L_log.txt: Log file that includes a summary table of model option settings for the run and integration performance data such as program runtime, number of integration steps, and number of function calls. The integration data is reported on the same interval as the other output files.

Miscellany

- To facilitate the modification of individual pieces of the equations of motion (gravitational torque, magnetic torque, orbit propagation, free motion, etc.), we rewrote the equations of motion subroutine to separate each of these parts into distinct modules. It is now much easier to isolate behavior and substitute specific components of the physical model.
- We cleaned up the data structure and expanded the use of global variables to allow for more efficient sharing of data within the model.
- Regrettably, time priorities kept us from inserting what would be a particularly useful feature. Currently, all the inputs and parameters are passed to the model via header files. This means that any changes to these values require the entire

program to be re-compiled. This is inefficient and robs the model of some of its potential utility. Better to dynamically load the input data from a separate file at runtime, making it much easier to run different scenarios with the same underlying physical model.

3.7.5 W02 Lageos Spin Model Software Package

Finally, we now briefly summarize the content of the W02 software package; the complete source code for the model is provided in an appendix to this document. The source files are organized a Dev-C++ Project in four modules: i) model control, ii) physical model, iii) numerical routines, and iv) optimization package. These are described as follows.

Model Control

This is the control center of the model. It includes the main "driver" program and evolution routines, input and output functionality, variable architecture, and project management features. The source files in this module are

Lmain.c: Contains the main program, the various driver routines, and a global dynamic memory allocation;

Lio.c: Contains all of the input (i.e., variable model parameter initializations) and output functionality of the W02 model;

Lincl.h: Common header file attached to every source file in the model; it includes common macros, header files (standard, GSL, and custom),

custom data structures, and Fortran subroutine interface macros; attached to every source file in the model;

Lprot.h: Header file containing subroutine prototypes for every function in the model; included in Lincl.h;

Lglob.h: Header file containing all global variable declarations; attached to Lmain.c;

Lextern.h: Header file containing "external" references to all global variables; attached to all source files except Lmain.c.

Physical Model

This module contains the actual dynamical modeling features of the W02 software package including the equations of motion, torque components, and parameter values. This is accomplished with the following source files:

Lderivs.c: Source code for the equations of motion, orbit propagation and torque computations;

Ligrf2000.f: Source code (Fortran) for the IGRF 2000 magnetic field model

Ltools.c: Various custom mathematics and astrodynamics utilities

Lparams.h: Header file contain all W02 control and physical parameters; this is the primary user interface and is well documented; included in Lincl.h

Numerical Routines

This component contains the external source numerical integration routines:

Ldop853.c: Source code for the DOPRI8 integration algorithm;

Ldop853.h: Header file for the DOPRI8 integration algorithm; attached to Ldop853.c;

Lnr_c.c: Source code for routines adapted from Numerical Recipes in C [ii]; includes BD, BS, and the numerical differentiation routine used by the parameter optimization package;

Lshode.f: Source code (Fortran) for the DE integration package.

Optimization Package

This package contains the implementation of the parameter optimization functionality. The main driver program diverts control to the optimization package, which then calls back to the central model to generate the data for the error function. The package is self-contained in the sense that all of the optimization specific subroutines, control parameters, input/output, and data are contained within this component. The package consists of

Lopt.c: Source code for the optimization subroutines including the optimization driver routine, the error function, the error gradient function, and optimization specific input/output functions;

Lopt.h: Header file containing Avizonis data, optimization control parameters and variables, and GSL header file includes for GSL functions contained only in the optimization package.

Compiler and Linker Options Summary

The project compiles to a Win32 console application (i.e., a command line executable). To obtain the most efficient code possible, the full optimizing capabilities of the GCC compiler are invoked using the compiler flags "-O3 -fexpensive-optimizations." Recalling the previous references, we summarize all the additional linker and compiler flags used in the construction of the executable:

- Compiler: -DHAVE_INLINE -std=gnu9x -O3 -fexpensive-optimizations
- Linker: -lgsl -lgslcblas -lg2c –lm

This fully describes our implementation of the W02 Lageos spin model from the software development perspective.

3.8 Summary

The issues involved with developing a comprehensive model of the Lageos spin dynamics are expansive. Using the H&W model as a baseline, we detailed all of the pertinent concerns. The physical model components were revisited in a bottom-up approach to ensure the best possible capture of dynamical effects. The effort led to a revision of the orbit module, the addition of the J_2 zonal harmonic term to the gravity

torque computation, and several 'tweaks' of the magnetic model. The analysis also suggested that a parameterized approach to representing the satellite was warranted. Therefore, an optimization capability was added to facilitate parameter tuning for optimal performance. Concerning numerical implementation, we presented efficient algorithms for each dynamical component of the model. More importantly, the numerical integration capabilities of the H&W model were improved upon by adding both a more efficient integrator and an integrator more suited to derivative function irregularities that may soon be an issue for Lageos. Finally, the software development of the model was explored to document the "behind the scenes" effort so that the model is more accessible for future use and revision. With the central thrust of the effort complete, we now look at some of the results and explore in more detail the issues raised along the way.

4 Results and Analysis

4.1 Overview

To this point, the primary emphasis has been on model development. In addition to bettering the predictive accuracy of the model, the detailed analysis uncovered some compelling ideas that merit consideration. With the construction complete, we now demonstrate the results, both in terms of exhibiting improved predictive performance and by using the model as a tool to study specific concerns. The presentation is, by nature, open-ended. We provide a sample of outputs that target the primary results with their immediate corollaries, and offer the model itself, anticipating its role in addressing new questions that emerge.

In seeking quantitative results, a fundamental issue arises. The true evolution of the Lageos spin state is unknown, hence the need for modeling efforts. This presents a quandary. If the truth is not known, how can predictive accuracy be verified? This question is answered in two parts. First, the comprehensive approach to refining the model is itself a claim to improvement; effects are implemented in more detail than they

were previously. In the absence of comparison data, this is often the sole basis for making such a claim. Second, a proxy for the unknown truth is used—the Avizonis data.

4.1.1 Avizonis 'Truth' Data

The empirical data provided by Avizonis [1997] and the process used to obtain it was described in Section 1.4.1. Two points are important: i) the data completely characterizes the spin state providing both spatial orientation and spin angular velocity and ii) the predictions are not exact, but rather, are the result of a statistical reduction of multiple candidate solutions suggested by the flash data. The former point allows the Avizonis data to fully comply with the role of truth surrogate. On the other hand, the latter point suggests that the data need not be taken too literally. In fact, in a reversal of roles, there is reason to believe the outputs of our model could be useful in tightening the Avizonis predictions.[1]

In all, Avizonis provides 29 individual data sets at epochs between September 1988 and October 1996. The predictions are concentrated in the 1992-93 timeframe, so we have focused on that interval for data comparison.[2] The reported spatial orientation (right ascension and declination of the spin axis) is the root-mean-square (RMS) minimum of

---

[1] This idea was mentioned in relation to Currie's present efforts to use Avizonis' approach (Currie, [2002]) where a quality a-priori spin state estimation is a necessity. While processing the older flash data was possible without such a contribution, the result would still likely benefit from a cooperative refinement.

[2] There are three early data points in 1988-89 that are too isolated to be of much use for prediction comparison. The five most recent data points from 1995-96 suffer a similar deficiency but are also subject to greater uncertainty due to the slowing spin rate and so are less reliable for comparative analysis.

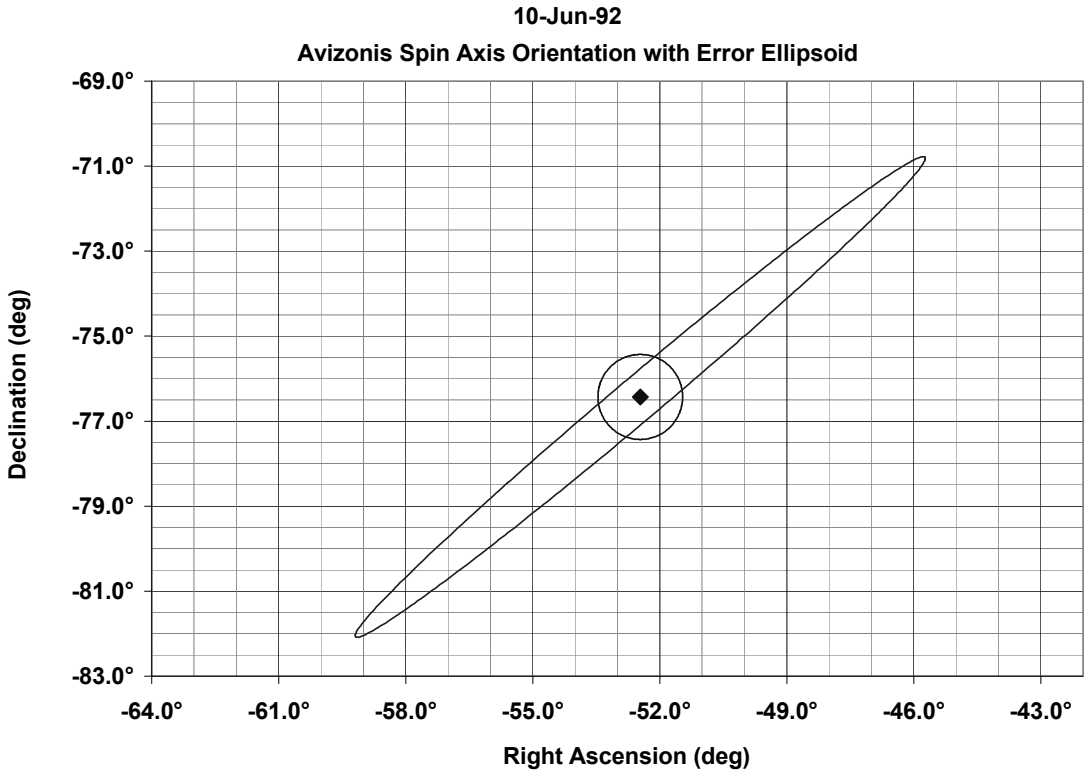

Figure 4.1 Sample Avizonis Lageos spin axis orientation solution with corresponding error ellipsoid
The spatial orientation for the Lageos satellite spin state on June 10, 1992 is shown (centered diamond) with the corresponding 1σ error ellipsoid superposed. The solution was determined by Avizonis from optical flash data. A unit sphere is also illustrated to provide a spatial benchmark.

the individual solutions. Also provided is an *error ellipsoid* corresponding to the statistical one-sigma (1σ) uncertainty in the result. An example for a specific data point is illustrated in Figure 4.1. Figure 4.2 depicts the complete set of 1992-93 Avizonis spin axis solutions; the error ellipsoids are also shown. Refer to Figure 3.5, on page 71, for the Avizonis spin angular velocity measurements.

In addition to providing a benchmark for comparative analysis, the Avizonis data is also used to supply the initial spin state for the numerical integration. The right ascension

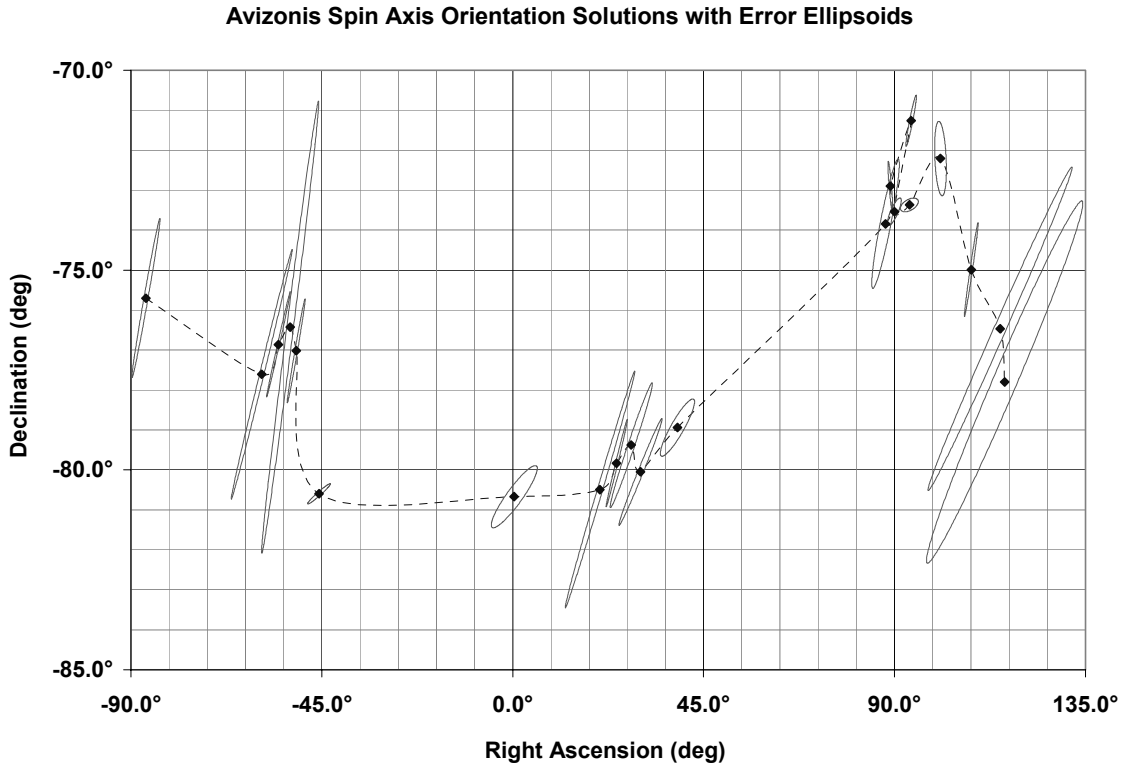

Figure 4.2 Lageos spin axis solutions with error ellipses from April 1992 to November 1993

Avizonis' spatial predictions for the Lageos spin axis are shown along with the corresponding error ellipses. The diamonds at the center of the ellipses indicate the RMS minimum. The time evolution of the data begins with the point on the very left edge of the plot; the sequence is indicated by the connecting dashed line. The scales on the axes differ causing a vertical elongation of the ellipsoids.

and declination have immediate correlation to the Euler angles ϕ and θ respectively. Moreover, symmetry allows the transverse axes to be arbitrarily specified, i.e., ψ_o can be any value we choose. Together, then, the initial values for the Euler angles based on a particular Avizonis data set (α, δ) are given by

$$\begin{aligned} \phi_o &= \alpha + 90° \\ \theta_o &= 90° - \delta \\ \psi_o &= 0° \end{aligned} \quad . \qquad 95$$

The initial rates for these angles derive from the Avizonis spin angular velocity ω. In the present analysis, the body and spin axes are assumed coupled so that

$$\dot{\psi}_o = \omega \,. \qquad 96$$

The true transverse rates are non-zero, but nevertheless, inconsequential; they are set to zero without loss of generality.

Given the potential error in the Avizonis data, some discrimination is warranted in the selection of a starting point. In particular, data sets with small error ellipsoids are preferred. The integration can be performed forward or backward in time, but intuition favors the former. Thus, initializing the integration near the beginning of the data set is also desirable. With these criteria, the H&W model chose to specify the initial conditions using the solution near –45° right ascension, which is based on optical flash observations collected on July 29, 1992 (designated 920729).[3] Table 4.1 provides the

Table 4.1 Avizonis data and Euler angle spin state from the July 29, 1992 (920729) data set

Avizonis Spin Axis Solution		
Right Ascension	α	–45.68°
Declination	δ	–80.60°
Spin Rate	ω	2.800 °/s
Initial Euler Angle Spin State		
Precession Angle	ϕ	44.32°
Nutation Angle	θ	170.60°
Spin Angle	ψ	0.00°
Precession Angle Rate	$\dot{\phi}$	0.000 °/s
Nutation Angle Rate	$\dot{\theta}$	0.000 °/s
Spin Angle Rate	$\dot{\psi}$	2.800 °/s

[3] Hereafter, it will be convenient to use this yymmdd format to reference specific Avizonis data points.

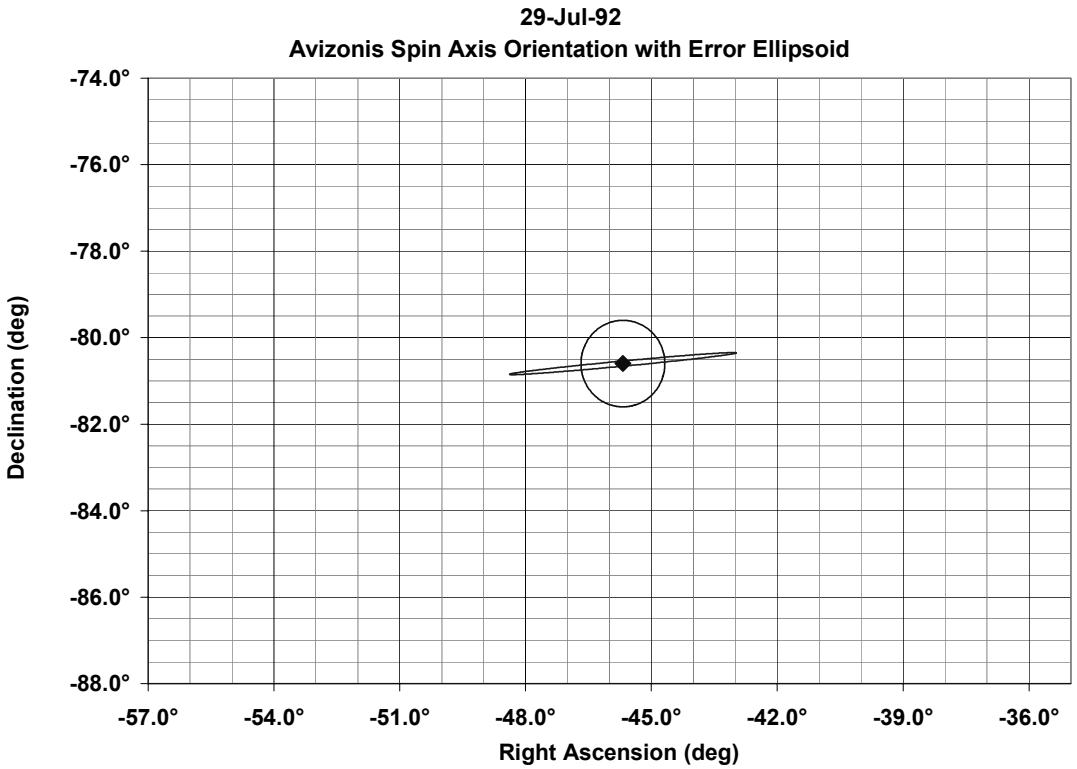

Figure 4.3 Avizonis Lageos spin axis solution for July 29, 1992

The 920729 Avizonis data is shown with axes scaled identically to Figure 4.2 to allow a direct visual comparison of the ellipsoid sizes. The small area of the 1σ ellipsoid suggests this point is a good candidate for initializing the model.

specific values corresponding to this data point as well as the resultant Euler angle spin state. A "zoomed-in" view of the 920729 data set is shown in Figure 4.3; the scale is set to match that of Figure 4.1 to illustrate the relative sizes of the error ellipsoids.

Almost all of the testing we performed during the development of the W02 model was also initialized with the 920729 data. This was done to allow direct comparison of new data with that from the H&W model. With the completed W02 model, however, data runs have been initialized using a number of different points. Table 4.2 provides the

Avizonis data for the starting values used most often in the analysis; although, only a subset of the data is presented here. To simplify the subsequent discussion, we refer to the initial conditions by the Avizonis set ID. Implicit in the reference is both the integration start time and the corresponding Euler angle spin state derived from the Avizonis data.

Table 4.2 Avizonis data sets used to initialize the model

Set ID (Date)	Right Ascension	Declination	Spin Rate
920406	–86.44°	–75.70°	3.076 °/s
920602	–55.22°	–76.86°	2.940 °/s
920729	–45.68°	–80.60°	2.800 °/s
920901	–0.36°	–80.67°	2.699 °/s
930428	88.98	–72.90°	2.158 °/s

One surprising discovery based on the results from the W02 model casts doubt on the quality of the 920729 data set; we now suspect it may be a *phantom* solution.[4]. The evidence will be apparent in the subsequent discussion, but suffice to say, that the point is a persistent outlier relative to the quality of the results at the other data sets.

4.2 General Results and Analysis

With the Avizonis data as a benchmark, the specific progress made with the W02 model is now analyzed. At the onset, two points are worth mentioning about the data presented. First, the parameters for the magnetic torque component are adjusted to ensure the spin

[4] Due to the geometries of the problem, the Avizonis method leads to several possible independent solutions, which are then filtered subject to specific criteria. While the process is sound, it is nevertheless possible that a phantom solution is taken instead of the true solution.

rate decay output by the model matches the empirical data. Therefore, unless explicitly stated otherwise, proper spin decay behavior is implicit in the results, and we dispense with redundant presentations of Figure 3.5. Second, in lieu of lengthy citations in the text of all model settings for each data run, only the important parameters that make the run unique will be stated. However, for repeatability, the runtime headers from the L_log.txt data file listing all model settings (see Section 3.7.4) are reported in Appendix A.

4.2.1 H&W Model Space-Time Tracking Error

We begin by recalling the remarks associated with Figure 1.1. Because the H&W model was incapable of producing results at specific times, the picture of the space-time correlation between the predicted spin state and the Avizonis data was incomplete. Figure 4.4 shows the same H&W[5] output as Figure 1.1 but with the correlated data points indicated.

The deceptive nature of the apparent spatial agreement between the model's output and the Avizonis data is clear once time is taken into account. The trend of the data appears reasonable, but the corresponding data points are often significantly separated. These differences are quantified in Table 4.3. The mean RSS error over the data set is 20.6°, and most of this is explained by poor right ascension tracking. Also note the extreme outlier of the group is the 920729 data point. This will be a persistent theme.

---

[5] To be precise, the data was generated using a 'beta' version of the W02 model that included targeted output time capability but retained the dynamical characteristics of the H&W model.

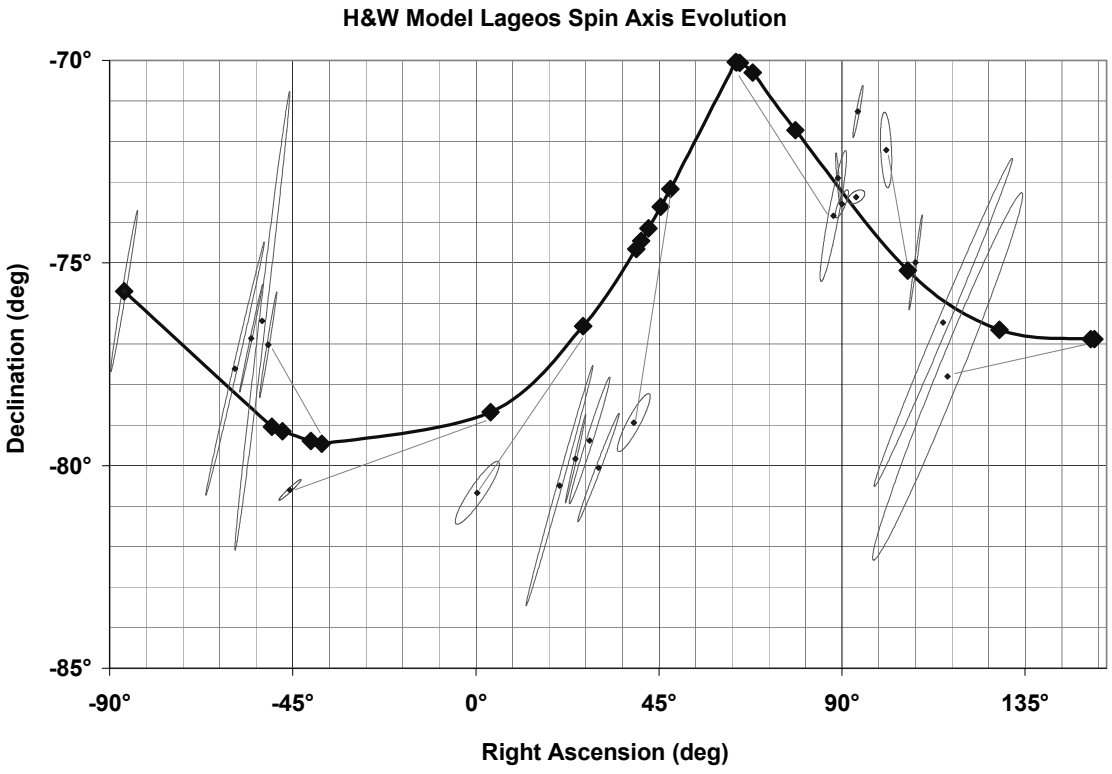

Figure 4.4 H&W model Lageos spin axis evolution with targeted output correlated to Avizonis data

The large diamonds indicate specific output times corresponding to Avizonis data sets. Light grey lines connect a few of the correlated data points to assist with the visualization. While the spatial trend of the data is qualitatively accurate, the space-time correlation is poor. The 920406 set provided the initial conditions for the data run.

Table 4.3 Spatial errors in the H&W model Lageos spin state propagation.
The raw errors for right ascension and declination are absolute values of the difference between model output and Avizonis data. Also shown is the root sum of squares (RSS) of right ascension and declination errors. The data exhibits very poor tracking performance by the H&W model, particularly in the right ascension angle. More than a quarter of the data points have an RSS error of 25° or worse.

Avizonis Data Set	Raw α Error	Raw δ Error	RSS Error
920406	0.00°	0.00°	0.00°
920530	9.02°	1.43°	9.13°
920602	7.65°	2.29°	7.99°
920610	11.89°	2.96°	12.25°
920613	13.20°	2.44°	13.43°
920729	49.31°	1.92°	49.34°
920901	26.03°	4.11°	26.36°
920929	18.88°	5.83°	19.76°
921002	16.11°	5.37°	16.98°
921007	14.48°	5.23°	15.40°
921015	15.33°	6.44°	16.63°
921023	9.02°	5.77°	10.71°
930424	24.04°	3.80°	24.34°
930428	24.89°	2.85°	25.05°
930507	29.08°	1.20°	29.11°
930602	21.88°	3.24°	22.12°
930717	14.91°	1.65°	15.00°
930915	5.34°	2.98°	6.11°
931016	20.64°	1.66°	20.70°
931111	36.24°	0.41°	36.24°
931113	36.05°	0.92°	36.06°

4.2.2 W02 Model Outputs

Turning our attention to the W02 model, we begin with the major result of this work. The primary goal for the W02 model was to improve upon the relatively poor space-time correlation exhibited above. Figure 4.5 depicts the W02 spin state propagation generated with parameters optimized for a best fit over the entire 1992-93 Avizonis data set using the 920406 data point for initialization. The *920406 global parameter set*[6] values are listed in Table 4.4.

Table 4.4 W02 optimized parameter values – the 920406 global parameter set

σ was fixed in the optimization because it does not provide a significant degree of independence from κ and a. f is a deprecated parameter so not modified in the optimization.

I_1	1.2798×10^{8} g cm^2
I_3	1.3070×10^{8} g cm^2
a	23.534 cm
σ	1.0×10^{17} s^{-1}
κ	2.4970
f	1.0

The achievement of the W02 model is immediately apparent. The predicted spin orientations are a much tighter fit to the central Avizonis data points. Moreover, where the predictions still miss the target, they are much more consistent with the error ellipsoids than the H&W model outputs. Together, these are profound results and represent a much stronger connection between theoretical and empirical modeling efforts than has previously been achieved.

---

[6] An optimization spanning the set of 1992-93 Avizonis data values is called a *global* optimization; the optimization is *local* if only a subset of the data is used. Optimized parameter sets will also be identified by the data point used as the initial value. The model parameters used to generate the data for Figure 4.5 are the *920406 global parameter set*.

Table 4.5 quantifies these results. The mean RSS error over the data set has been reduced by 70% to 6.0°. The RSS errors are also much more consistent from point to point than for the H&W model. Excluding the extreme outlier, which is once again the 920729 data point, the standard deviation of the remaining RSS errors has been reduced by a factor of four.

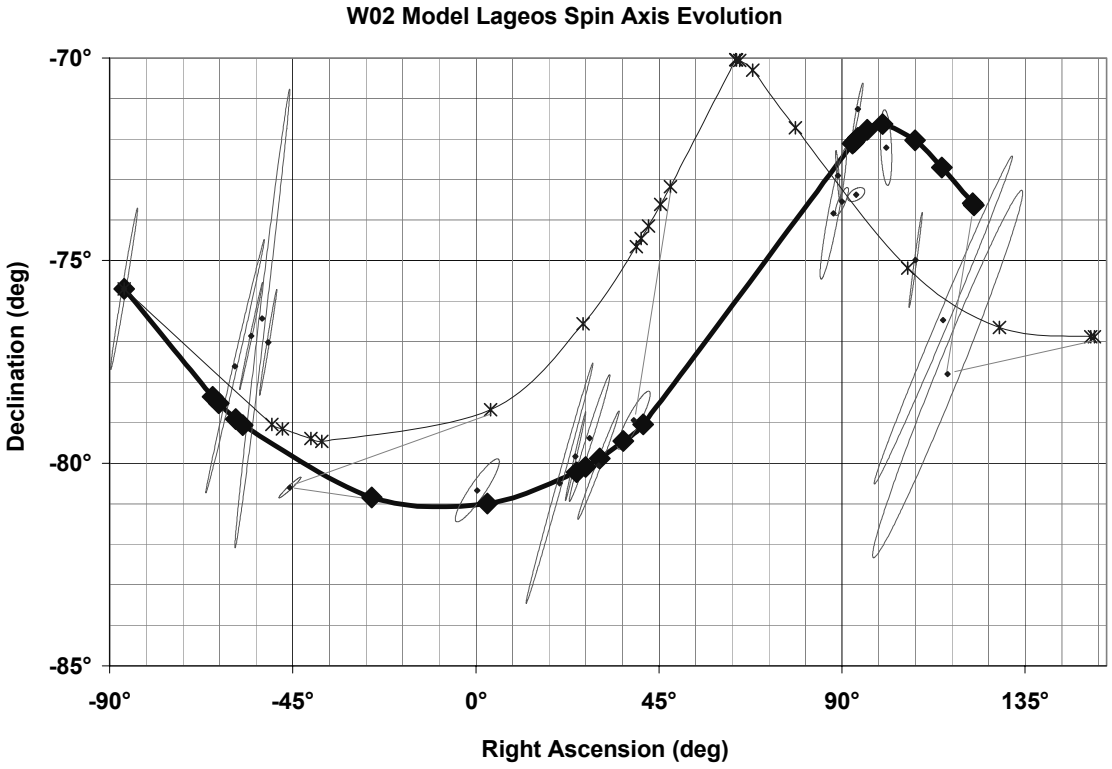

Figure 4.5 Comparison of W02 and H&W models' predicted Lageos spin states with Avizonis Data
The W02 model output (large diamonds with heavy solid line) shows much better tracking with the Avizonis data than the H&W model (crosses with light solid line). Correlations with the Avizonis data at a few of the data points are indicated by light grey lines. The run initialized on the 920406 data point and the 920406 global parameters were used in the model.

Table 4.5 Spatial errors in the W02 model Lageos spin state propagation

Error data reported for the W02 model is in the same format as that of Table 4.3. Excepting the outlier at 920729, the data exhibits a dramatic improvement over the H&W model with RSS spatial errors of 8° or less.

Avizonis Data Set	Raw α Error	Raw δ Error	RSS Error
920406	0.00°	0.00°	0.00°
920530	5.49°	0.75°	5.54°
920602	7.91°	1.66°	8.08°
920610	6.48°	2.48°	6.94°
920613	6.23°	2.04°	6.56°
920729	20.06°	0.24°	20.06°
920901	2.48°	0.32°	2.50°
920929	4.25°	0.27°	4.26°
921002	2.47°	0.26°	2.48°
921007	2.46°	0.50°	2.51°
921015	6.15°	0.59°	6.18°
921023	2.27°	0.11°	2.27°
930424	4.72°	1.73°	5.03°
930428	4.02°	0.83°	4.11°
930507	0.04°	0.71°	0.71°
930602	6.18°	1.77°	6.43°
930717	6.50°	1.74°	6.73°
930915	7.07°	0.18°	7.07°
931016	6.41°	2.29°	6.80°
931111	7.22°	2.89°	7.78°
931113	6.47°	4.17°	7.70°

4.2.3 Observations

A number of observations can now be made about the W02 results. First, the data represents a specific customized case. A complete exploration of the issue requires varying both the starting point and the optimization technique. We have performed some of this analysis, and the results are interspersed on the subsequent pages starting with Figure 4.6. However, the combinations are endless and one can quickly get lost in minutia. Therefore, we will discuss the central ideas and remind that the model is available for further investigation of specific concerns.

Second, an objection might be raised that tuning the model to fit the data doesn't necessarily imply quality performance beyond the benchmark data set. A valid issue but also one that we have already addressed at length. In spite of every best effort, the model is still only an abstraction of the true system. It is, therefore, both contrary to reason and no less arbitrary to hold any particular part of the model or its parameters as absolute. As long as the tuning retains its connection to the physical system,[7] it is not only justified, but also even to be expected. Investigating various data runs as suggested in the preceding paragraph is a good way to explore potential sensitivity to the issue.

[7] That is, the physical implementation and the corresponding parameters do not deviate too far from nominal.

Third, while the improved performance of the W02 model over its predecessor is substantial, there remains opportunity for further progress. This can be explored on a number of different levels. For one, further refinements to the physical model may still be necessary. Then again, some of the remaining error might be partly explained by the uncertainty in the Avizonis data. That is, the model is actually performing better than the

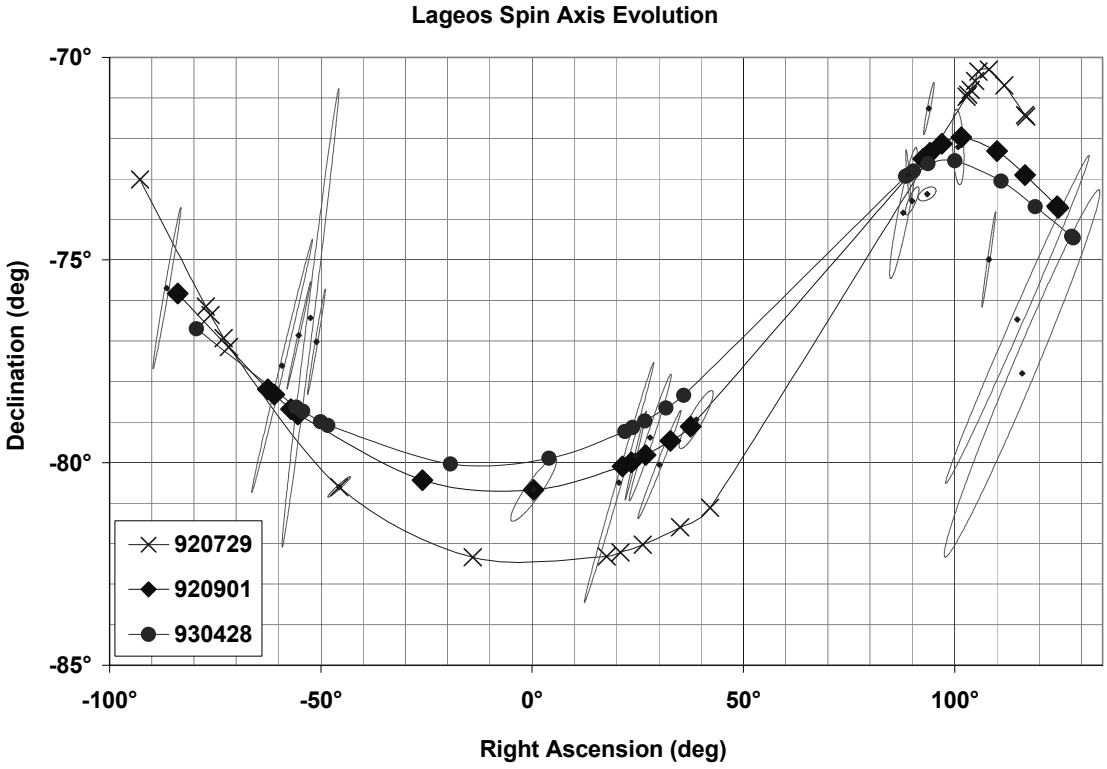

Figure 4.6 W02 with 920406 global parameters; multiple data runs using different initial conditions
The plot features data from a few W02 model runs all using different start times but the same 920406 global parameter set. Integration was performed forward and backward to span the Avizonis data set. The data exhibit tracking performance similar to the baseline case in Figure 4.5. In fact, this holds for additional runs we performed but omit from the graph to avoid clutter. This suggests some degree of initial-state independence in the global optimization parameter set.

error computations suggest because the comparison points also deviate from the true spin axis orientation. This observation has already led us to suggest the cooperative role of our model in refining the empirical data.

Finally, parameter optimization is itself an open-ended process that depends heavily on the characterization of error. Certainly, the optimization approach we implemented can be further improved, thereby yielding even better results. For example, weighting the error calculation to account for the proximity to the error ellipsoids and allowing the initialization point to drift within its error ellipse might yield further substantial gains. Excluding outliers, such as the 920729 data point, may likely also improve the performance of the optimization over the data set as a whole.

Local Optimization Example

Figure 4.6 shows the W02 output using the 920406 global parameter set for several different sets of initial conditions. The relative consistency among the results provides some measure of confidence that the output isn't hyper-sensitive to the tuning.[8] The results might have been partly expected, however, because the optimization was global. This raises a question of how a local parameter set might perform over the broader array of data.

---

[8] Spin state propagations starting from different epochs will necessarily have some variance in behavior. A set of optimized parameters for a given interval and epoch (e.g., the 920406 global parameter set) will not usually be ideal for any other start time even if the optimization covers the same span of data. To claim general applicability of the model, however, a particular optimal parameter set should not lead to wildly divergent behavior when used with different initial data points.

To explore this idea, we performed a local optimization using a set of six data points spanning about two months time beginning with the initial conditions specified by the 920901 data point. The resulting local parameter set is provided in Table 4.6. Figure 4.7 shows the data run corresponding to this 920901 local parameter set. The figure caption provides some additional insights.

Table 4.6 W02 optimized parameter values – a 920901 local parameter set

I_1	1.2870×10^{8} g cm^2
I_3	1.3144×10^{8} g cm^2
a	23.549 cm
σ	1.0×10^{17} s^{-1}
κ	2.500
f	1.0

Similar to the case for Figure 4.6, a number of data runs were generated using the 920901 local parameter set and different initial conditions. The outputs of some of these runs are provided in Figure 4.8. The results are encouraging. First, even though the parameters were locally optimized over only a handful of data points, the integration with the 920901 initial condition performs quite well globally. In fact, as Table 4.7 shows, the individual error results are similar to those of the global optimization. Second, the data runs (not all are shown in Figure 4.8) using different initial conditions exhibit the same kind of general agreement with the empirical data that was observed in Figure 4.6. The tracking isn't quite as tight as the global parameter cases, but that is to be expected for local parameters. The result suggests even locally optimized parameters demonstrate a degree of initial state independence, adding to the credibility of the W02 model.

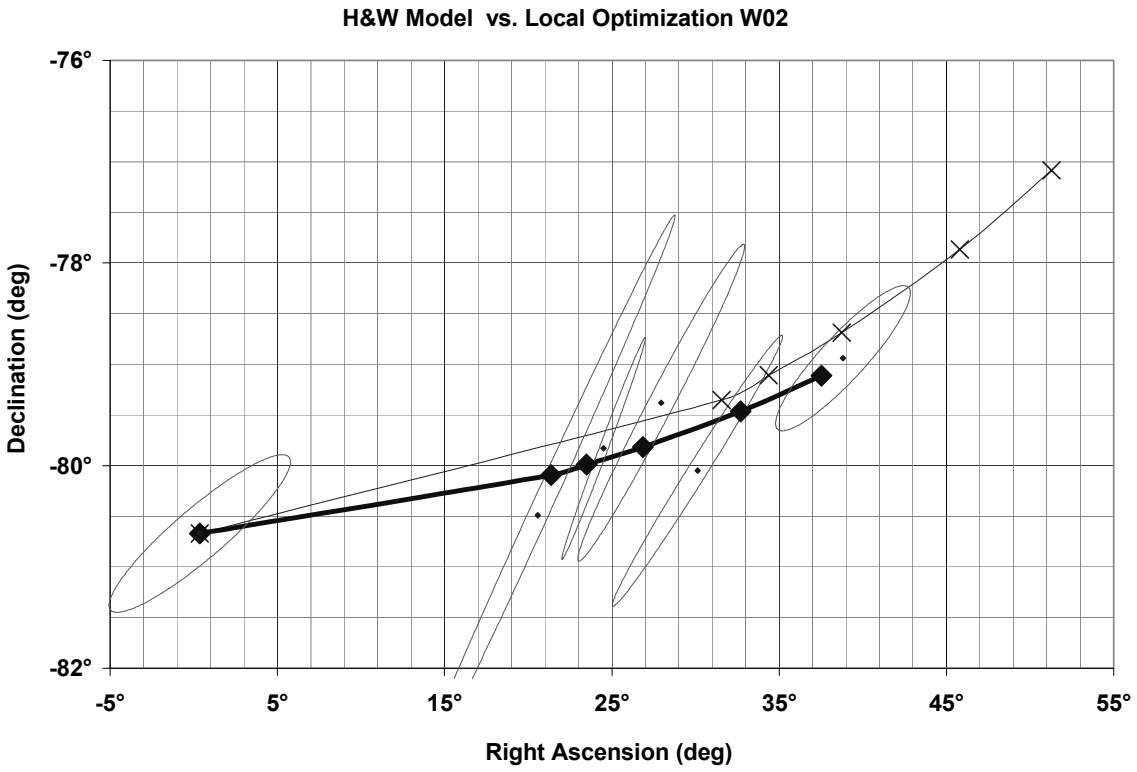

Figure 4.7 W02 model local optimization Lageos spin state performance compared to H&W model

The plot depicts output from the H&W model (crosses with light solid line) and the W02 model (diamonds with heavy solid line) both initialized on the 920901 data point. The parameters used in the W02 model are a 920901 local parameter set obtained by performing a local optimization over the Avizonis data points shown in the graph. The superb agreement between the W02 results and the empirical data suggests the optimization approach is capable of producing extremely high quality predictions over short intervals and lends credence to the idea of using the model in an iterative predictor-corrector approach to refine observation based spin state estimates.

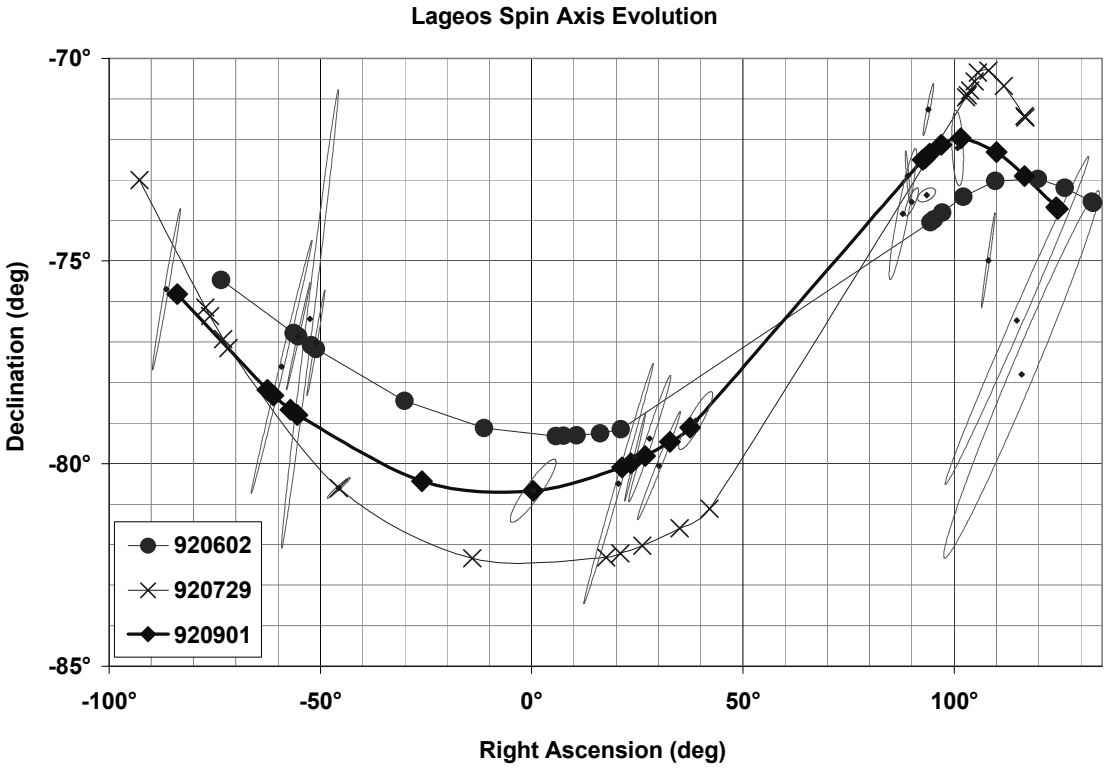

Figure 4.8 W02 with 920901 local parameters; multiple data runs using different initial conditions

As was the case in Figure 4.6, the chart includes only a sample of the data runs we performed. The results and analysis are similar to the situation described in Figure 4.6. They suggest even a locally optimized parameter set provides greatly improved global performance and a measure of initial-state independence.

Table 4.7 Spatial errors in the W02 model Lageos spin state propagation using a 920901 local parameter set

Error data reported for the W02 model is in the same format as that of Table 4.3. The relatively poor quality at the 920729 data point is again apparent; otherwise, the data shows performance similar to that of the global case in Table 4.5.

Avizonis Data Set	Raw α Error	Raw δ Error	RSS Error
920406	2.57°	0.12°	2.58°
920530	3.36°	0.57°	3.41°
920602	5.85°	1.46°	6.02°
920610	4.60°	2.24°	5.11°
920613	4.43°	1.79°	4.77°
920729	19.72°	0.17°	19.72°
920901	0.00°	0.00°	0.00°
920929	0.79°	0.40°	0.89°
921002	1.03°	0.16°	1.04°
921007	1.09°	0.44°	1.17°
921015	2.57°	0.59°	2.64°
921023	1.28°	0.17°	1.29°
930424	4.73°	1.34°	4.92°
930428	4.12°	0.44°	4.14°
930507	0.24°	1.09°	1.12°
930602	6.99°	1.41°	7.13°
930717	8.07°	1.40°	8.19°
930915	9.21°	0.10°	9.21°
931016	8.61°	2.09°	8.86°
931111	9.33°	2.80°	9.74°
931113	8.57°	4.09°	9.50°

In spite of the preceding general arguments to the contrary, one picture that emerges over multiple runs with differing parameter sets and initial conditions is that some data points are better suited as initial conditions than others. The case against the 920729 data point has already been made, but a similar idea can be seen in Figure 4.8 with the 920602 point. We believe this suggests something about the data rather than the model. The two data points just cited illustrate the two possible explanations. The 920729 data point, as we have already mentioned, may be a false solution and should be revisited. A different possibility exists for the 920602 data point, where the true solution is likely not the RMS minimum but rather another point within the uncertainty region. This can be seen visually in Figure 4.8 where the lower tip of the 920602 1σ error ellipse (third from the left) very nearly touches the very good solution curve corresponding to the 920901 initial condition. Both of these points are further confirmation that the W02 model can play a productive role in helping to refine the empirically determined spin state solutions.

4.3 Additional Investigations

With the major result established, we now undertake a preliminary discussion of specific issues that have arisen. This treatment is not meant to be the final word, but rather, to elevate an awareness of the ideas. Three topics in particular are explored: i) the sensitivity of the results to small errors in the values of the principal moments, ii) the timing of the spin and body axis decoupling, and iii) the suggestion that the spin axis solutions may be spatially inverted from the true orientation.

4.3.1 Sensitivity to Small Changes in Principal Moments

One of the more notable discoveries of our efforts is the sensitivity of the spin state propagation to small changes in the satellite parameters. This is particularly true with respect to the satellite principal moments (see Figure 4.9), a point to which we alluded earlier. Previous modeling efforts apparently did not consider the possibility that small deviations in the principal moments would make much of a difference [Kheyfets, private

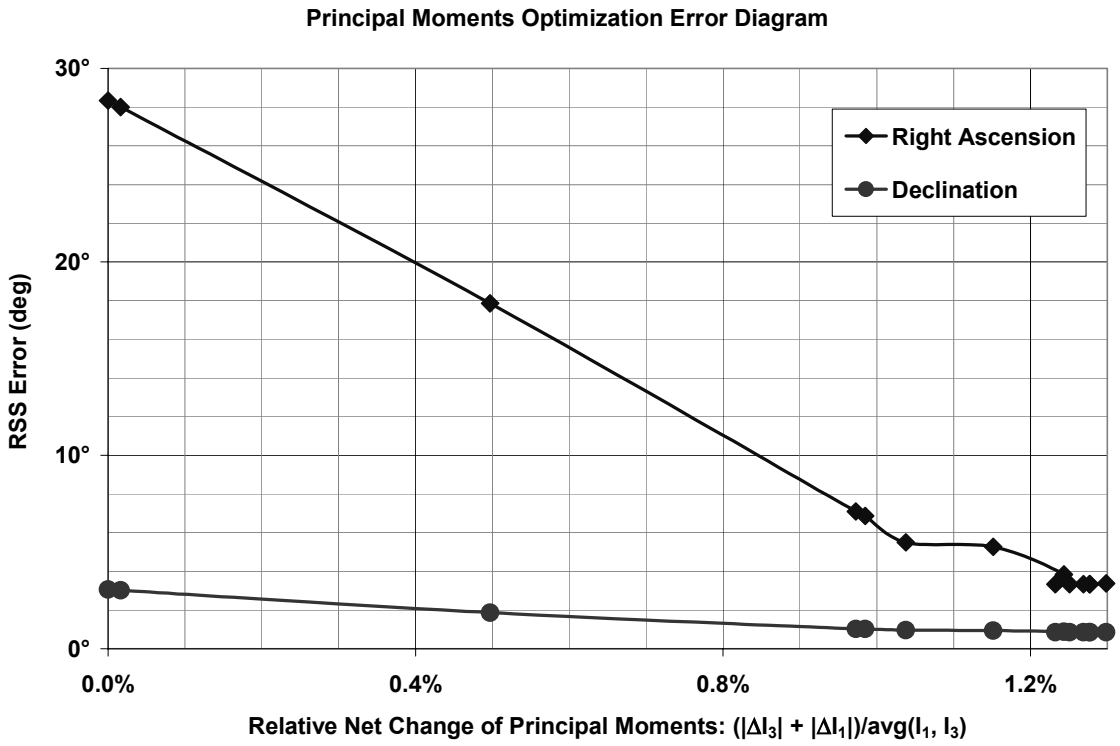

Figure 4.9 Spatial RSS errors as a function of the relative net change of the principal moments

The RSS errors for right ascension and declination are shown for a local optimization in which only the principal moments were allowed to vary. The setup was otherwise the same as the case in Figure 4.7. The principal moments were seeded with their nominal values $I_3 = 1.314\times10^8$ and $I_1 = 1.271\times10^8$ (g cm^2). The data exhibits a disproportionally large reduction of spatial error (90% in α and 70% in δ) in response to a relatively small change in the principal moments (net 1.3%) of the satellite.

communication]. Moreover, unlike the magnetic torque model where the abstraction mandates a parameterized approach for the values describing the satellite properties, the principal moments have a solid basis in physical reality. Therefore, departure from the empirically determined values listed in Table 2.1 requires some justification. Exaggerated response to small changes in model parameters also raises the question of whether the effect is physically valid or simply an artifact of numerical modeling (e.g., poor conditioning). Both issues are examined to demonstrate the effect is a valid physical response and that small departures from the nominal values are reasonable.

There are compelling reasons to treat the principal moments as flexible parameters in the model. The first was argued in Section 4.2.3; the generalizations inherent to the modeling process implicitly authorize a parameterized view as long as the connection to the physical system is maintained. Nevertheless, there are also sound physical arguments for assuming the principal moments may deviate from the nominal values. For one, a statistical error in the measured values of the moments is likely. Whether a confidence interval was reported with the original data is undetermined (see footnote 6 on page 39), but the values are reported with only three significant digits so at least ~0.5% error can be presumed. The thermoelastic deformation described in Section 3.3.1 also perturbs the values of the principal moments. The magnitude is uncertain but the effect becomes more pronounced as the spin rate decays (slower spin means a greater temperature imbalance across the satellite). Therefore, while we don't recommend deviating too far

from the empirical values for the principal moments, the net combined relative change represented in Figure 4.9 seems entirely reasonable.

The analytical justification for a heightened sensitivity to small changes in the principal moments was suggested at the end of Section 3.4.1. Both the free response portion of the equations of motion

$$\begin{aligned}
\ddot{\theta}_{fr} &= -\frac{\dot{\varphi}\sin\theta}{I_1}\left[(I_3 - I_1)\dot{\varphi}\cos\theta + I_3\dot{\psi}\right] \\
\ddot{\varphi}_{fr} &= \frac{\dot{\theta}}{I_1\sin\theta}\left[(I_3 - 2I_1)\dot{\varphi}\cos\theta + I_3\dot{\psi}\right] \\
\ddot{\psi}_{fr} &= -\frac{\dot{\theta}}{I_1\sin\theta}\left[\left((I_3 - I_1)\dot{\varphi}\cos\theta + I_3\dot{\psi}\right)\cos\theta - I_1\dot{\varphi}\right]
\end{aligned} \tag{14}$$

and the expressions for the gravitational torques

$$\boldsymbol{N}_g = \frac{3\mu_e}{r^3}(I_3 - I_1)\begin{bmatrix} \rho_2\rho_3 \\ -\rho_1\rho_3 \\ 0 \end{bmatrix} \tag{40}$$

and

$$\Delta\boldsymbol{N}_g = \frac{3\mu_e J_2 R_e^2}{2r^5}(I_3 - I_1)\begin{bmatrix} 5(1-7\sin^2\lambda_c)\rho_2\rho_3 - 10\sin\lambda_c(\rho_2 T_{33}^{\mathrm{EB}} + \rho_3 T_{23}^{\mathrm{EB}}) - 2T_{23}^{\mathrm{EB}}T_{33}^{\mathrm{EB}} \\ -\left[5(1-7\sin^2\lambda_c)\rho_1\rho_3 - 10\sin\lambda_c(\rho_1 T_{33}^{\mathrm{EB}} + \rho_3 T_{13}^{\mathrm{EB}}) - 2T_{13}^{\mathrm{EB}}T_{33}^{\mathrm{EB}}\right] \\ 0 \end{bmatrix} \tag{43}$$

are heavily influenced by the *difference* ($I_3 - I_1$) between the axial and transverse principal moments. Small relative changes in the principal moments correlate to large relative changes in their difference, as illustrated in Table 4.8. While the table only shows the results for changes to I_1, the effect is similar for I_3. It can be seen, in

Table 4.8 Impact of small relative changes to the principal moments

The table shows how a small relative change in I_1 relates to a large change in the difference between the principal moments, $I_3 - I_1$. The axial component is held fixed at $I_3 = 1.314\times10^8$ to simplify the illustration, but the correlation applies to both principal moments without loss of generality.

Transverse Principal Moment (I_1)	Relative Change from Nominal	Difference (I_3–I_1)	Relative Change of Difference
1.271e+8	0.00%	4.300e+6	0.0%
1.277e+8	0.50%	3.665e+6	14.8%
1.284e+8	1.00%	3.029e+6	29.6%
1.290e+8	1.50%	2.394e+6	44.3%
1.296e+8	2.00%	1.758e+6	59.1%

particular, that small changes to both values that cooperatively close the gap between them will result in a large overall impact. Therefore, the effect is built into the physical system; it is not an artifact of the simulation.

4.3.2 Spin-Body Axis Decoupling

There is anticipation in the Lageos literature for significant dynamical changes to the Lageos spin state when the spin frequency of the satellite reaches the orbital frequency (Habib et al [1994] call this the *spin-orbit resonance*). Among the more important effects is a decoupling of the spin and body axes. The resulting spin motion complicates efforts to study the behavior of the Lageos satellite. Specifically, we recall Currie's effort to apply the Avizonis method to recently obtained optical data sets. With decoupled motion, a fundamental assumption of the method no longer holds, so the results cannot be supplied with any measure of confidence.

Table 4.9 Projected Lageos spin rates

The table shows historical and future spin rates for the Lageos satellites. The projections are based on the exponential decay with a 780 JD half-life. Because the magnetic field beat frequency is twice the orbit frequency, the spin-body axis decoupling will begin 780 JD sooner than most projections suggest. This result is confirmed by the multiple frequency correction in our magnetic torque module.

Date (JD2K)	Lageos Spin Rate	Description
-4110	9.572 °/s	Avizonis Data
-2826	3.076 °/s	Avizonis Data
-2439	2.158 °/s	Avizonis Data
-1755	1.216 °/s	Avizonis Data
0	0.251 °/s	Projection: J2000
1743	0.053 °/s	Projection: 2x Orbit Frequency
1826	0.049 °/s	Projection: 2005
2523	0.027 °/s	Projection: Orbit Frequency
3653	0.010 °/s	Projection: 2010

It is therefore important to secure an accurate prediction of the timing of the spin-body axis decoupling. We raised the issue (see Section 3.5.4) that most references place the event's beginning too far in the future. In particular, the projections fail to account for the pseudo-symmetry of the magnetic field that generates to a field beat-frequency twice that of the orbital motion (recall Figure 3.9).

Table 4.9 extrapolates the Lageos spin rate based on the empirically determined decay half-life of 780 JD. The anticipated matching of the satellite and orbit angular velocities does not occur until late November 2006, but we predict the "spin-orbit resonance" dynamics to occur more than two years earlier, in October 2004. These results are validated by the data exhibited in Figure 4.10 on page 172. Additional commentary accompanies the charts.

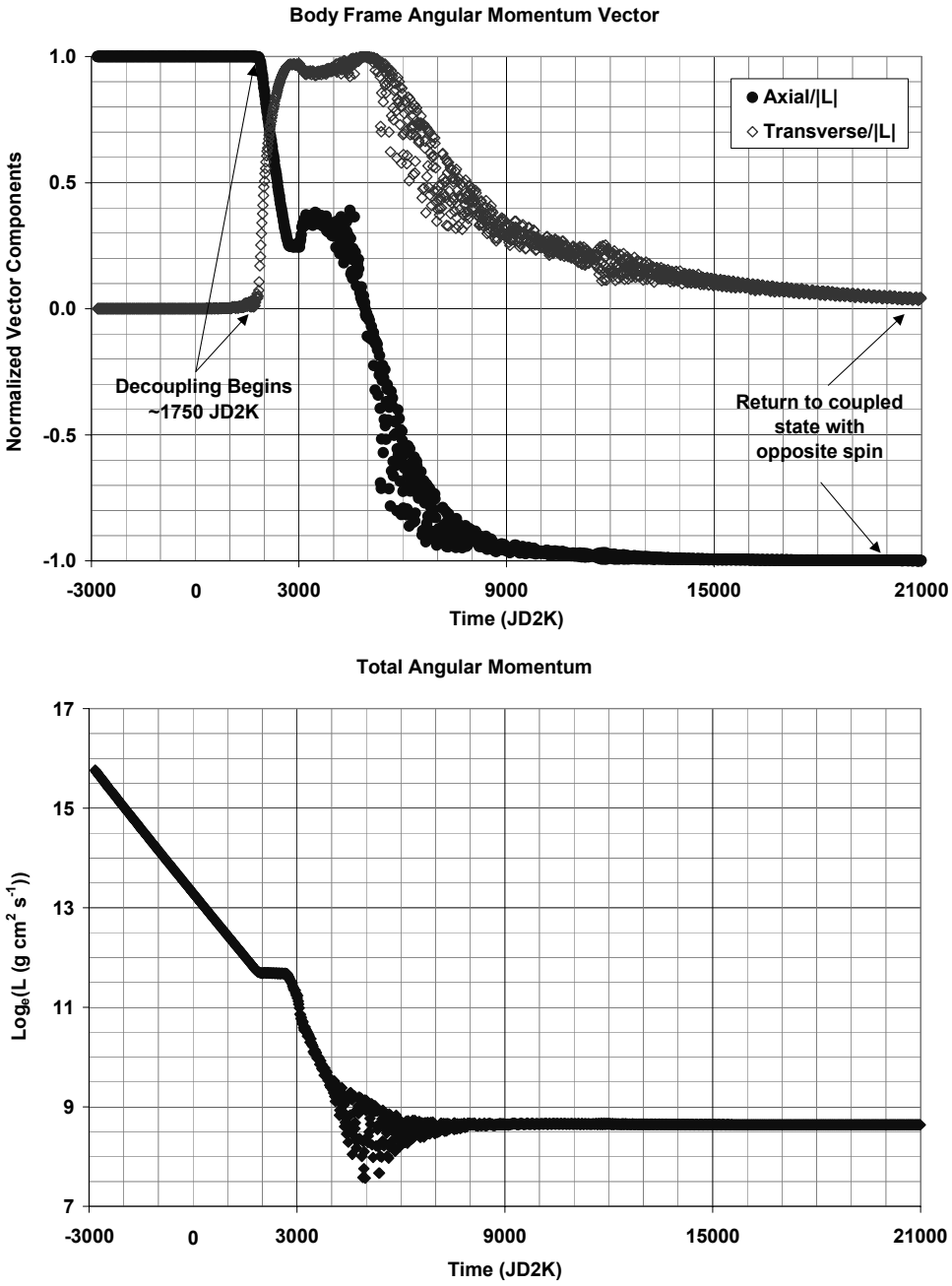

Figure 4.10 Long term evolution of the Lageos spin angular momentum

The normalized body frame components of the spin angular momentum vector (top) agree with the predicted body-spin decoupling as seen by the transfer of angular momentum to the transverse component. Simultaneously, the total angular momentum (bottom) briefly halts its decay. Later, the total angular momentum settles to a constant value, and the satellite returns to a body-spin coupled state but with the rotation in the opposite sense. The data was generated using the 920406 global parameter set.

4.3.3 Spatially Inverted Pole

Finally, we briefly comment on an earlier issue. There is uncertainty as to the original orientation of the Lageos spin vector following the spin-up at orbit insertion. The spatial location of the initial spin axis was determined by Rubincam [1987] to have right ascension and declination respectively of 313° and 68°. The uncertainty relates to the direction of spin along this axis—the "northern" solution favored by Rubincam places the spin axis at $\{\alpha = 313°, \delta = 68°\}$[9] while others (e.g., Farinella & Vokrouhlicky [1996], Barlier et al [1996]) have suggested the spatially inverted "southern" solution at $\{\alpha = 133°, \delta = -68°\}$. The latter groups also claim that the dynamics are not invariant under this transformation.

Adding to the discussion, Avizonis [1997] appeals to a continuity argument and concludes that the pole cannot have flipped since the initial solution. This implies that the initial choice of either the northern or southern solution must persist in subsequent data through the present (though this will change once spin-body axis decoupling takes hold). Ironically, Avizonis credits Rubincam for the pole solution influencing the reduction of the optical data, but the spin orientation results he provides actually show the southern solution was used.

By extension, we too have implemented the southern bias in our work. This was simply a result of utilizing the Avizonis data. Whether the southern solution is

[9] Rotation expressed in the customary counterclockwise direction.

authenticated as the correct choice is undetermined. However, based more on an intuition supplied by countless data runs than on hard analysis, the picture appears to be mixed. On the one hand, given the strong performance demonstrated by the W02 model in the preceding sections, it seems unlikely that the wrong data was targeted. On the other, some of the responses observed during the development stages of the W02 model might be better explained by a spatially inverted data set. This latter view would put us in the northern hemisphere with Rubincam and against the majority of recent opinions on the issue.

The additions to the magnetic torque model, in particular, provided some interesting results. For example, the now deprecated oblateness factor was inserted to allow the cylindrical core to be approximated by a slightly flat (equatorial radius larger than polar radius) oblate spheroid. However, better results were achieved for an elongated spheroid. There was also a similar hint that the directional scaling of the coefficients of magnetization might be more productive if the roles of the parallel and normal directions were reversed. This is consistent with the response of the oblateness parameter. Perhaps these are valid responses that hint at an inverted data set.

The issue remains open, but it may not be particularly important. The W02 model uses the southern convention inherited from the Avizonis data, but if it turns out the northern solution is correct, an appropriate set of parameters can be easily derived using the included optimization package.

Summary

Of the three ideas just mentioned, the most impacting is the system's sensitivity to perturbations of the principal moments. This motivates a closer look at the issue as a high priority (next section). Spin-body axis decoupling is an inevitable event but understanding the timing is essential to proper analysis of optical data sets. The possibility of a spatially inverted data set is mostly a curiosity. The W02 Lageos spin model can easily adapt to either convention; it may even be useful in helping to resolve the issue.

4.4 Future Work

There is no finality to a project of this scope. In spite of significant accomplishments, several opportunities exist to further improve upon this work. Many of these ideas have already been suggested throughout the text. We summarize here and refer back to the appropriate sections for more detail.

Equations of Motion

- We chose the Euler angle characterization (Section 2.2.2) for two reasons: legacy with the earlier version of the model and the intuitive characterization of the spin state. However, a future revision may want to consider implementing the quaternion (or similar non-singular) formulation to avoid the numerical trap caused by the

artificial singularity of the Euler angle approach. As an added benefit, quaternions provide a boost in efficiency.

- Alternately, the singularity can also be treated by inserting a second inertial reference frame that is offset from the first by a simple rotation of the nutation angle (θ). This approach is somewhat more cumbersome but retains the intuition of the Euler angle formulation.

Physical Model

- Given the sensitivity of the system's response to small changes in the principal moments of the satellite, the potential uncertainties in these values must be better addressed. In particular, the role of thermoelastic deformation in perturbing the principal moments needs to be evaluated in greater detail. Because the instantaneous thermal properties of the satellite depend on the spin state (rate and orientation), there is an implicit time dependence of the thermoelastic deformation. Depending on the magnitude of the effect, the principal moments also become time dependent parameters rather than fixed constants. It is entirely feasible that a significant portion of the residual error might be squeezed out if this variance is taken into account.
- The magnetic torque module also continues to require attention. The multiple frequency adaptation (Section 3.5.4) is a "first cut" solution that will benefit from further analysis; although, it is encouraging that the model generated the expected spin-body axis decoupling behavior (Section 4.3.2). The alternate approach to the

multiple frequency problem we mentioned also bears further consideration. Finally, because the Earth's rotation frequency is just one order of magnitude smaller than the 2x orbit frequency, it may also be prudent to include this effect in the multiple frequency computation.

- We are pleased with the directional scaling of the magnetization coefficients (also Section 3.5.4), both in concept and in application. As a further refinement, however, the scaling might alter the direction cosine approach to account for the non-smooth cylindrical edge condition in the transition between body axis normal and parallel directions.

Optimization

- The results of the preceding sections provide strong confirmation of the parameterized approach. It should be reiterated, however, that the parameter optimization is highly dependent on the definition of the error equation to be minimized (Section 4.2.3). Accordingly, a computation that better utilizes the statistical error information associated with the benchmark data may provide yet another dramatic improvement to the predictive capability of the model. Specifically, the spatial errors might be computed at each point in terms of components along the major and minor axes of the error ellipsoid and scaled, respectively, by the semi-major and semi-minor axis values.

- A further refinement of the optimization process is suggested by the residual initial-state dependence of the parameter sets (end of Section 4.2.3). Rather than fixing the initial orientation in the data runs, the starting values for right ascension and declination could also be inserted as optimization parameters. This would completely remove the bias imposed by a specific initial state.

Data Analysis

- Briefly, the inverted pole issue (Section 4.3.3) can be evaluated by analyzing the performance of the model against the mirror solutions for the Avizonis data.
- Finally, as we have mentioned, the existing empirically determined data sets can be refined using a cooperative iteration of the optimized model performance and the Avizonis reduction method. This would provide a better location of the likely spatial orientation within the statistical error bounds of a given data point and help to filter out phantom solutions in favor of previously discarded results.

Summary

Of the ideas just listed, three are most impacting: i) reformulating the equations of motion; ii) addressing the thermoelastic effects; and iii) refining the error equation in the optimization process. If these are accomplished, there is every reason to believe that another advance proportional to the leap from the H&W model to our own W02 model is possible.

4.5 Conclusion

Context

In the decades since Lageos I began its orbital life, spacecraft have become increasingly complicated. With strange geometries, mechanical inner-workings, and autonomous attitude control, the modern breed of satellites make a poor orbital benchmarks due to the uncertainties surrounding the corresponding perturbations. Accordingly, much of the 'mainstream' literature on spacecraft torques treat the issues only to the extent that they enlighten a discussion on corrective control mechanisms. In this realm, the environmental effects on passive satellites seem an antiquated topic.

And yet, it is precisely because of the lack of 'modern' features that Lageos has surfaced as an important, space-based laboratory for the study of a vast array of geodynamic phenomenon. Because of the extremely precise orbit position determination available for Lageos, the satellite is an excellent candidate for the study of small orbit perturbing phenomena, including the general relativistic gravitomagnetic force. Proper isolation of these effects require a specific understanding of Lageos' spin dynamics so that surface thermal forces (e.g., Yarkovsky drag) can be isolated. This has led to a host of modeling efforts from both empirical and theoretical perspectives.

Accomplishments

Unfortunately, none of the modeling efforts to date can provide the degree of predictive accuracy required by the above applications. The work we present here goes a long way

toward closing the gap in providing high quality predictive results for the Lageos spin state. Using the most general of the existing theoretical models as a baseline, we accomplished three specific goals.

First, we performed a comprehensive review of the general problem. This led to specific enhancements of the baseline model and a tightening of the confidence in the overall approach. Specifically:

- The orbit model was revised to correct a secular error in the angular position tracking and incorporate the modest effects of the elliptical motion.
- The gravitational torque module now includes the effects of the Earth's J_2 zonal spherical harmonic.
- The simplistic dipole approximation for the geomagnetic field has been replaced with the IGRF geomagnetic field model; a 10 stage spherical harmonic expansion of the Earth's magnetic field. Balancing precision and efficiency, we use the first three terms of the series.
- The magnetic torque model has been enhanced to provide a scaled directional dependence of the magnetization coefficients based on the orientation of the spacecraft with respect to the magnetic field. In addition, the torque is computed for the field frequency due both to the satellite spin and the orbital motion. The results are linearly superposed to provide a first order approximation in response to the challenging multiple frequency problem.

As a corollary to this bottom-up approach, the present document represents perhaps the most comprehensive encyclopedia for the dynamical motion of the Lageos satellite.

Second, we overhauled the model from a numerical and software perspective to generate a useful tool for present work and future adaptation. In particular, the numerical integration capabilities were enhanced to provide both more efficient and more adaptive options. In addition, the software architecture was restructured to make use of powerful constructs of the programming language and incorporate useful third party packages. However, the most important accomplishment of the numerical revision was the addition of the parameter optimization capability. This feature introduces a profound flexibility into the Lageos spin model that enables new possible applications, including an expanded role in cooperatively refining the empirical spin state solutions.

Finally, we demonstrated a dramatic improvement in predictive capability, establishing for the first time that quantitative results are possible in the Lageos spin dynamics problem with the theoretical modeling approach. Consequently, there now exists a much stronger connection between theoretical and empirical efforts. Specifically, we showed a 70% reduction in the mean RSS error of the spin state predictions for the Avizonis data set spanning 1992-93 and near perfect tracking for a more localized set. Moreover, the analysis has helped identify future modifications that will enable further improvements to these results.

In the process of achieving these goals, a number of ideas that heretofore had apparently gone unnoticed were exposed. These include the impact of small changes to

the principal moments of the satellite, the related role of thermoelastic deformation, and a faster than anticipated spin-body axes decoupling due to the 2x orbit frequency oscillation of the magnetic field. Our investigation of these issues was preliminary but suggests implications that merit future consideration.

In short, we accomplished our objectives, and more. The revised Lageos spin model is established as a powerful theoretical tool for investigating the dynamical behavior of the satellite.

List of References

Literature

Ascher, U. M. and L. R. Petzold (1998).
Computer Methods for Ordinary Differential Equations and Differential Algebraic Equations, Society for Industrial and Applied Mathematics (SIAM).

Avizonis, P. V. (1997).
"Remote Sensing of the LAGEOS-I Spin-Axis and Image Processing for Advanced Optical Systems," Doctoral Thesis, Department of Physics, University of Maryland at College Park.

Avizonis, P.V. (2002).
Technology Development, Exponent, Inc. (private communication)

Barlier, F., P. Farinella, and D. Vokrouhlicky (1996).
"The Rotation of LAGEOS and its Long-term Semi-major Axis Decay: A Self-consistent Solution," *Journal of Geophysical Research*, 101, 17861

Bertotti & Iess (1991).
"The Rotation of Lageos," *Journal of Geophysical Research*, 96-B2, 2431-2440.

Booth, M., J. Davies, M. Galassi, B. Gough, G. Jungman, F. Rossi, and J. Theiler (2001).
GNU Scientific Library Reference Manual, 1st Ed for GSL Version 1.0, published under the GNU Free Documentation License.

Bulirsch, R. and J. Stoer (1966).
"Numerical treatment of ordinary differential equations by extrapolation methods," *Numerical Mathematics*, 8, 1-13.

Bulirsch, R. and J. Stoer (1993).
Introduction to Numerical Analysis, 2nd Ed., Springer-Verlag.

Burley, J. C. (2002).
Using and Porting GNU Fortran, Free Software Foundation.

Campbell (1996).
"Problem with the "MAGNETIC" Pole Locations on Global Charts," *Eos*, 77:36, 345-347

Campbell, S. L. (1997).
Department of Mathematics, North Carolina State University (private communication).

Chobotov, V. A. (1991).
Spacecraft Attitude Dynamics, Krieger Publishing

Ciufolini, I. (1986).
"Measurement of the Lense-Thirring drag effect on LAGEOS and another high altitude laser ranged satellite," *Physics Review Letters*, 56, 278-281.

Ciufolini, I. (2002).
"Test of general relativity: 1995-2002 measurement of frame-dragging," General Relativity–Quantum Cosmology Abstract gr-qc/0209109, *Proceedings of Physics in Collision Conference*.

Ciufolini, I., D. Lucchesi, F. Vespe, and F. Chieppa (1997),
"Detection of Lense-Thirring Effect Due to Earth's Spin," General Relativity-Quantum Cosmology Abstract gr-qc/9704065, publication status unknown

Ciufolini, I., and J. A. Wheeler (1995).
Gravitation and Inertia, Princeton University Press.

Currie, D. (2002).
Department of Physics, University of Maryland at College Park (private communication).

Danby, J. M. A. (1992).
Fundamentals of Celestial Mechanics, William-Dell, Inc.

Dormand, J. R., and P. J. Prince (1981).
"High order embedded Runge-Kutta formulae," *Journal of Computational & Applied Mathematics*, 7, 67-75.

Dueflhard, P. (1983).
"Order and Stepsize Control in Extrapolation Methods," Numerische Mathematik, 41, 399-422.

Edwards, C. H. and D. E. Penney (1985).
Elementary Differential Equations with Applications, Prentice-Hall, Inc.

Farinella, P., and D. Lucchesi (1991).
"Optical Properties of the Earth's Surface and Long-Term Perturbations of LAGEOS' Orbit," submitted to *Journal of Geophysical Research*.

Farinella P., and D. Vokrouhlicky (1996).
"Thermal force effects on slowly rotating artificial satellites. I. Solar heating," *Planetary Space Sciences*, 44, 1551

Featherstone, W. E. (1996).
"A Compendium of Earth Constants Relevant to Australian Geodetic Science," *Geomatics Research Australia*, 64.

Flannery, B. P., W. H. Press, S. A. Teukolsky, W. T. Vetterling (1992).
Numerical Recipes in C, The Art of Scientific Computing, 2nd Ed., Cambridge University Press.

Gill, E. and O. Montenbruck (2000).
Satellite Orbits-Models, Methods, Applications, Springer-Verlag.

Goldstein, H. (1980).
Classical Mechanics, 2nd Ed.., Addison-Wesley Publishing

Gordon, M.K. and L. F. Shampine (1975).
Computer Solutions of Ordinary Differential Equations, Freeman and Company

Gremaud, P. A. (2000).
Department of Mathematics, North Carolina State University (MA 780 Numerical Analysis coursenotes).

Haberman, R. (1987).
Elementary Applied Differential Equations, 2nd Ed., Prentice-Hall, Inc.

Habib, S., D. Holz, A. Kheyfets, R. A. Matzner, W. A. Miller, and B. W. Tolman (1994).
"Spin Dynamics of the LAGEOS satellite in support of a measurement of the Earth's gravitomagnetism," *Physical Review D*, 50:10, 6068-6079.

Hairer, E., S.P. Norsett and G. Wanner (1993)
Solving Ordinary Differential Equations I, Nonstiff Problems, 2nd Ed., Springer-Verlag.

Hoots, F. R. and R L. Roehrich (1980).
”Models for Propagation of NORAD Element Sets,” *Spacetrack Report Number 3*.

Hughes, P. C. (1986).
Spacecraft Attitude Dynamics, John Wiley & Sons, Inc.

Iorio, L. (2001).
“Nongravitational perturbations on the mean anomaly of LAGEOS type satellites and the gravitomagnetic clock effect,” General Relativity-Quantum Cosmology Abstract gr-qc/7057, publication status unknown.

Iorio, L. (2000).
“Satellite gravitational orbital perturbations and the gravitomagnetic clock effect,” General Relativity-Quantum Cosmology Abstract gr-qc/007014, submitted to *International Journal of Modern Physics*, D.

Jackson, J. D. (1975).
Classical Electrodynamics, John Wiley & Sons, Inc.

Johnson, C. W., C. A. Lundquist, and J. L. Zurasky (1976).
”The Lageos Satellite,” conference paper, The XXVIIth Congress of the International Astronautical Federation.

Kelso, T. S. (1995-6).
“Orbital Coordinate Systems, Parts I–III,” *Satellite Times*, 2:1–3 (available online at [G]).

Kelso, T. S. (1998).
“Frequently Asked Questions: Two-Line Element Set Format,” *Satellite Times*, 4:3 (available online at [G]).

Kheyfets, A. (1992).
“Spin Dynamics of the LAGEOS Satellite,” report, Air Force Summer Research Program, publication status unknown.

Kheyfets, A. (1993).
“Spin Evolution of the LAGEOS Satellite,” report, Air Force Summer Research Program, publication status unknown.

Kheyfets, A. (2002).
Department of Mathematics, North Carolina State University (private communication).

Landau, L. D. and E. M. Lifshitz (1984).
Electrodynamics of Continuous Media, Pergamon.

Meyer, C. (2000).
Matrix Analysis and Applied Linear Algebra, Society for Industrial and Applied Mathematics (SIAM).

Ohanian, H. C. (1985).
Physics, Volume I & II, W. W. Norton & Company.

Perko, L. (1991).
Differential Equations and Dynamical Systems, 2nd Ed., Springer-Verlag.

Roy, A. E. (1988).
Orbital Motion, Adam Hilger

Rubincam, D. P. (1987).
"Lageos orbit decay due to infrared radiation from Earth," *Journal of Geophysical Research*, 92, 1287-1294.

Rubincam, D. P. (1990).
"Drag on the Lageos Satellite," *Journal of Geophysical Research*, 95, 4881-4886.

Rubincam, D. P. (1990).
"The Lageos Along-Track Acceleration: A Review," conference paper, First William Fairbank Meeting on Relativistic Gravitational Experiments in Space, Rome.

Shabana, A. A. (2001).
Computational Dynamics, 2nd Ed., John Wiley & Sons, Inc.

Stallman, R. M. (2002).
Using the GNU Compiler Collection, Free Software Foundation.

Wiesel, W. E. (1989).
Spaceflight Dynamics, McGraw-Hill, Inc.

Williams, S. (1997).
"An Improved Model of the Spin Dynamics of the Lageos Satellite," Master's Thesis, Department of Mathematics, North Carolina State University.

Electronic Media

[A] Encyclopedia Astronautica, Spacecraft Index-LAGEOS, http://www.astronautix.com/craft/lageos.htm

[B] JPL Mission and Spacecraft Library, "Lageos 1 & 2 Quicklook," http://msl.jpl.nasa.gov/QuickLooks/lageosQL.html

[C] Looking at Earth from Space—40+ Years of NASA Earth Science History, "The Lageos Program," http://www.earth.nasa.gov/history/lageos/lageos.html

[D] The International Laser Ranging Service (ILRS) Satellite Missions, "Lageos 1, 2," http://ilrs.gsfc.nasa.gov/satellite_missions/list_of_satellites/lageos.html

[E] Small Satellites Home Page, Satellite Photos, http://www.ee.surrey.ac.uk/SSC/SSHP/PIX/LAGEOS1.GIF

[F] NORAD Two-Line Element Set Archives, LAGEOS-1, assembled by Dr. T. S. Kelso, http://celestrak.com/NORAD/archives/

[G] Satellite Times – Computers and Satellites, Dr. T. S. Kelso, http://celestrak.com/columns/index.shtml

[H] NASA Spacelink – Laser Geodynamic Satellites, http://spacelink.nasa.gov/NASA.Projects/Earth.Science/Land/Laser.Geodynamics.Satellites/

[I] The National Institute of Standards and Technology (NIST) Reference on Constants, Units, and Uncertainty, http://physics.nist.gov/cuu/index.html

[J] The International Earth Rotation Service (IERS) Conventions http://www.iers.org/iers/products/conv/

[K] The International Association of Geodesy (IAG) Geodesist's Handbook – Parameters of Common Relevance of Astronomy, Geodesy, and Geodynamics, http://www.gfy.ku.dk/~iag/HB2000/part4/groten.htm

[L] Earth's Magnetic Field, http://denali.gsfc.nasa.gov/research/mag_field/conrad/explain.html

[M] NSSDC Model Web – External (T96) and Internal Geomagnetic Field Model Parameters, http://nssdc.gsfc.nasa.gov/space/cgm/t96.html

[N] Spenvis Magnetic Field Models Background Information – "Dipole Approximations of the Geomagnetic Field" http://www.spenvis.oma.be/spenvis/help/background/magfield/cd.html

[O] GEOPACK, http://nssdc.gsfc.nasa.gov/space/model/magnetos/data-based/geopack.html

[P] International Geomagnetic Reference Field – Epoch 2000 Revision Of The IGRF for 2000 – 2005, http://www.ngdc.noaa.gov/IAGA/wg8/igrf.html

[Q] EC Material Properties – Conductivity of Metals, http://www.cnde.iastate.edu/ncce/EC_CC/Sec.7.2/Sec.7.2.html

[R] AzoM.com – The A to Z of Materials, http://www.azom.com/default.asp

[S] Hairer Fortran and Matlab Codes, http://www.unige.ch/math/folks/hairer/software.html

[T] NIST Guide to Available Mathematical Software (GAMS), http://gams.nist.gov/

[U] GAMS Module ODE in ODE, http://gams.nist.gov/serve.cgi/Module/ODE/ODE/9853/

[V] GNU Scientific Library (GSL), http://sources.redhat.com/gsl/

[W] GNU Win32 Project – Win32 ports of GNU Tools, http://gnuwin32.sourceforge.net/

[X] The Dev-C++ Resource Site, http://devcpp.everfloyd.com/index.html

[Y] Bloodshed Software – Providing free software to the internet community, http://www.bloodshed.net/

[Z] MinGW – Minimalist GNU for Windows Homepage, http://www.mingw.org/index.shtml

[AA] GCC – GNU Compiler Collection Hompage, http://gcc.gnu.org/

List of References

[BB] GCC Online Documentation,
http://gcc.gnu.org/onlinedocs/

[CC] Language Standards Supported by GCC,
http://gcc.gnu.org/onlinedocs/gcc/Standards.html#Standards

[DD] GNU's Not Unix – Free software foundation,
http://www.gnu.org/home.html

[EE] C and C++ Main Page References and Links,
http://home.att.net/~jackklein/c/c_main.html

[FF] Comeau Computing Tech Talk C/C++ FAQ,
http://www.comeaucomputing.com/techtalk/

[GG] Comeau Computing Tech Talk C99 FAQ,
http://www.comeaucomputing.com/techtalk/c99/

[HH] Using C and C++ with Fortran,
http://www.math.utah.edu/software/c-with-fortran.html

[II] Mixing Fortran and C – Collected Slides,
http://owen.sj.ca.us/rkowen/howto/slides/FandC/ALLF.html

[JJ] Netlib Repository – f2c package,
http://www.netlib.org/f2c/

[KK] Generating skeletons and prototypes with f2c,
http://gcc.gnu.org/onlinedocs/g77/f2c-Skeletons-and-Prototypes.html#f2c%20Skeletons%20and%20Prototypes

Numerical Packages

i. GEOPACK Library.
Reference: [O]
FORTRAN subroutines for magnetospheric modeling studies, including the current (IGRF) and past (DGRF) internal field models, a group of routines for transformations between various coordinate systems, and a field line tracer.

ii. Numerical Recipes in C
Reference: Flannery et al
C subroutines for a breadth of numerical programming applications; routines for numerical differentiation and numerical integration are directly or indirectly utilized.

iii. Hairer Codes for Nonstiff Differential Equations
Reference: [S], Hairer et al
Fortran and C subroutines for the numerical integration of non-stiff ordinary differential equations;

iv. GAMS ODE Module
Reference: [U], Gordon & Shampine
Fortran subroutines implementing the Shampine variable order variable step ABM PECE numerical integration method

v. GNU Scientific Library (GSL)
Reference: [V], [W], Booth et al [2001]
A collection of routines for numerical computing written in strict ANSI C and distributed under the GNU General Public License (free for non-proprietary use). The present GSL package is from the GNUWin32 Project, which provides a Win32 port of GNU tools.

vi. Dev-C++
Reference: [X], [Y], [Z], [AA]
A full-featured Integrated Development Environment (IDE) for the C/C++ programming language. It uses the Mingw port of GCC (GNU Compiler Collection) as it's compiler. It creates native console or GUI Win32 executables.

Appendices

Appendix A – Data Run Summary Headers

The W02 Lageos Spin Model generates a runtime header documenting all model settings for each data run (see L_log.txt description in Section 3.7.4). The runtime headers corresponding to data reported in the text are listed here with references to the locations the data was presented. For more information on the contents of the runtime header, see the Lparams.h file listed with the code.

Data Run for Figure 1.1

```
*****************************************************************************************************
*                                Lageos Spin Dynamics Model Version 2.7                             *
*                                                                                                   *
*    Date: 30 Nov 2002      Start Time: 19:13:59      Stop Time: 19:18:28      Total Time:   4:29.0    *
* ------------------------------------------------------------------------------------------------- *
* Run Description  > Baseline Data Run                                                              *
*                  > Input Source = Program Defaults                                                *
* Integrator Ctrl  > Driver = lageos_spin_de (variable order/variable step Adams PECE method       *
*                  > RTOL = 1.0e-008;  ATOL = 2.2e-016                                              *
*                  > MaxStepSize = 100.0s; MaxInternalTimeValue = 1.0e+006                          *
* Physical Const   > GM = 3.986004418e+020;  C = 2.99792458e+010                                    *
* Earth Magnetic   > Dipole Moment = 7.8115998e+025;  Pole Location = -71.406820 lon, 10.704433 co-lat *
* Orbit Params     > a = 1227119200.;  [e = 0.0];  i = 109.840460;  [w = 0.0];  [M = M+w]           *
*                  > RAAN = {109.051226,    0.3425558366}                                           *
*                  > M+w  = {319.491478, 2298.9786567857}                                           *
* Satellite        > Moments: I1 = 1.271e+008;  I3 = 1.314e+008                                     *
*                  > Metallic Core: radius = 26.35;  effective conductivity = 1.098e+017            *
* ICs (angle,rate) > theta = (165.70, 1e-016);  phi = (3.56, 1e-016);  psi = (0.00, 3.0764)         *
*                  > epoch = -2826.333206 (JD2K)                                                    *
* Units are cgs & degrees; some values implicit/not directly used depending on integration method   *
*****************************************************************************************************
```

Data Run for Figure 4.4 and Table 4.3

```
*****************************************************************************************************
*                              Lageos Spin Dynamics Model Version 2.7                                   *
*                                                                                                      *
*    Date: 12 Dec 2002      Start Time: 17:11:07      Stop Time: 17:13:28      Total Time:   2:20.5    *
* ---------------------------------------------------------------------------------------------------*
* Run Description  > H&W Model with targeted output times                                              *
*                  > Input Source = Program Defaults                                                   *
* Integrator Ctrl  > Driver = lageos_spin_de (variable order/variable step Adams PECE method          *
*                  > RTOL = 1.0e-008;  ATOL = 2.2e-016                                                 *
*                  > MaxStepSize = 100.0s; MaxInternalTimeValue = 1.0e+006                             *
* Physical Const   > GM = 3.986004418e+020;  C = 2.99792458e+010                                       *
* Earth Magnetic   > Dipole Moment = 7.8115998e+025;  Pole Location = -71.406820 lon, 10.704433 co-lat *
* Orbit Params     > a = 1227119200.;  [e = 0.0];  i = 109.840460;  [w = 0.0];  [M = M+w]              *
*                  > RAAN = {109.051226,    0.3425558366}                                              *
*                  > M+w  = {319.491478, 2298.9786567857}                                              *
* Satellite        > Moments: I1 = 1.271e+008;  I3 = 1.314e+008                                        *
*                  > Metallic Core: radius = 26.35;  effective conductivity = 1.098e+017               *
* ICs (angle,rate) > theta = (165.70, 1e-016);  phi = (3.56, 1e-016);  psi = (0.00, 3.0764)            *
*                  > epoch = -2826.333206 (JD2K)                                                       *
* Units are cgs & degrees; some values implicit/not directly used depending on integration method      *
*****************************************************************************************************
```

Data Run for Figure 4.5 and Table 4.5

```
*****************************************************************************************************
*                              Lageos Spin Dynamics Model Version 5.0                                   *
*                                                                                                      *
*    Date: 12 Dec 2002      Start Time: 17:57:45      Stop Time: 18:02:28      Total Time:   4:43.0    *
* ---------------------------------------------------------------------------------------------------*
* Run Description  > Data Analysis-Global Optimization Performance                                     *
*                  > Input Source = Program Defaults                                                   *
* Integrator Ctrl  > Driver = lageos_spin_de (variable order/variable step Adams PECE method          *
*                  > RTOL = 1.00e-8;  ATOL = 2.22e-16                                                  *
*                  > MaxStepSize = 100.0s;   MaxInternalTimeValue = 1.00e+7                            *
* Physical Const   > C = 2.99792458e+10;  RE = 6.3781366e+8;  GM = 3.986004418e+20;  J2 = 1.0826359e-3 *
* Earth Models     > Gravity gradient includes 1st nonspherical geopotential term (J2)                  *
*                  > Magnetic field generated using IGRF2000 spherical harmonic terms up to order 3    *
* Orbit Params     > a = 1227119174.;  e = 0.0044319;  i = 109.84188;  [w = (M+w) - M];                 *
*                  > RAAN     = { 109.051226,     0.34255584}                                          *
*                  > M+w Quad = { 319.420422,  2298.97906233, 1.77236871e-7}                           *
*                  > M Quad   = { 107.570967,  2299.19307317, 1.70332257e-7}                           *
* Satellite        > Moments: I1 = 1.2798e+8;  I3 = 1.3070e+8                                          *
*                  > MagTrq: OrbFrq=1  R(eq)=23.534  flat=0.00%  MCScl=2.50  MCShl=0.00  sig=1.0000e+17*
* ICs (angle,rate) > theta = (165.70, 1.0e-016);  phi = (3.56, 1.0e-016);  psi = (0.00, 3.0764)        *
*                  > epoch = -2826.333206 (JD2K)                                                       *
* Units are cgs & degrees; some values implicit/not directly used depending on integration method      *
*****************************************************************************************************
```

Note: the runs for Figure 4.6 use the same parameters as above with different initial conditions. The three starting points represented are from the 920729, 920901, and 930428 data sets and the corresponding initial states are derived from Table 4.2 in the manner described in the text.

Data Runs for Figure 4.7 and Figure 4.8

```
*****************************************************************************************************
*                              Lageos Spin Dynamics Model Version 5.0                                 *
*                                                                                                     *
*    Date: 13 Dec 2002      Start Time: 21:00:36      Stop Time: 21:04:19      Total Time:   3:43.4    *
* ----------------------------------------------------------------------------------------------------*
* Run Description  > Data Analysis                                                                    *
*                  > Input Source = Program Defaults                                                  *
* Integrator Ctrl  > Driver = lageos_spin_de (variable order/variable step Adams PECE method         *
*                  > RTOL = 1.00e-8;  ATOL = 2.22e-16                                                 *
*                  > MaxStepSize = 100.0s;   MaxInternalTimeValue = 1.00e+7                           *
* Physical Const   > C = 2.99792458e+10;  RE = 6.3781366e+8;  GM = 3.986004418e+20;  J2 = 1.0826359e-3 *
* Earth Models     > Gravity gradient includes 1st nonspherical geopotential term (J2)                *
*                  > Magnetic field generated using IGRF2000 spherical harmonic terms up to order 3   *
* Orbit Params     > a = 1227119174.;  e = 0.0044319;  i = 109.84188;  [w = (M+w) - M];                *
*                  > RAAN     = { 109.051226,     0.34255584}                                         *
*                  > M+w Quad = { 319.420422,  2298.97906233, 1.77236871e-7}                          *
*                  > M Quad   = { 107.570967,  2299.19307317, 1.70332257e-7}                          *
* Satellite        > Moments: I1 = 1.2870e+8;  I3 = 1.3144e+8                                         *
*                  > MagTrq: OrbFrq=1  R(eq)=23.549  flat=0.00%  MCScl=2.50  MCShl=0.00  sig=1.0000e+17*
* ICs (angle,rate) > theta = (170.67, 1.0e-016);  phi = (90.36, 1.0e-016);  psi = (0.00, 2.69865)     *
*                  > epoch (start) = -2678.446690 JD2K;    stop = -2240.446528 JD2K                   *
* Units are cgs & degrees; some values implicit/not directly used depending on integration method     *
*****************************************************************************************************

*****************************************************************************************************
*                              Lageos Spin Dynamics Model Version 2.7                                 *
*                                                                                                     *
*    Date: 13 Dec 2002      Start Time: 20:44:33      Stop Time: 20:47:13      Total Time:   2:40.1    *
* ----------------------------------------------------------------------------------------------------*
* Run Description  > H&W Model with targeted output times                                             *
*                  > Input Source = Program Defaults                                                  *
* Integrator Ctrl  > Driver = lageos_spin_de (variable order/variable step Adams PECE method         *
*                  > RTOL = 1.0e-008;  ATOL = 2.2e-016                                                *
*                  > MaxStepSize = 100.0s; MaxInternalTimeValue = 1.0e+006                            *
* Physical Const   > GM = 3.986004418e+020;  C = 2.99792458e+010                                      *
* Earth Magnetic   > Dipole Moment = 7.8115998e+025;  Pole Location = -71.406820 lon, 10.704433 co-lat *
* Orbit Params     > a = 1227119200.;  [e = 0.0];  i = 109.840460;  [w = 0.0];  [M = M+w]              *
*                  > RAAN = {109.051226,    0.3425558366}                                             *
*                  > M+w  = {319.491478, 2298.9786567857}                                             *
* Satellite        > Moments: I1 = 1.271e+008;  I3 = 1.314e+008                                       *
*                  > Metallic Core: radius = 26.35;  effective conductivity = 1.098e+017              *
* ICs (angle,rate) > theta = (170.67, 1e-016);  phi = (90.36, 1e-016);  psi = (0.00, 2.69865)         *
*                  > epoch = -2678.446690 (JD2K)                                                      *
* Units are cgs & degrees; some values implicit/not directly used depending on integration method     *
*****************************************************************************************************
```

Note: the additional W02 runs for Figure 4.8 use the same parameters as the Lageos 5.0 case above but with different initial conditions. The three starting points represented are from the 920602, 920729, and 920901 data sets and the corresponding initial states are derived from Table 4.2 in the manner described in the text.

Data Runs for Figure 4.9

```
*****************************************************************************************************
*                              Lageos Spin Dynamics Model Version 4.4b                              *
*                                                                                                   *
*    Date: 30 Nov 2002      Start Time: 01:14:35      Stop Time: 01:14:35      Total Time:   0:00.0    *
* ------------------------------------------------------------------------------------------------- *
* Run Description  > Local Optimization - Principal moments only                                    *
*                  > Input Source = Program Defaults                                                *
* Integrator Ctrl  > Driver = lageos_spin_de (variable order/variable step Adams PECE method       *
*                  > RTOL = 1.00e-8;  ATOL = 2.22e-16                                               *
*                  > MaxStepSize = 100.0s;   MaxInternalTimeValue = 5.00e+6                         *
* Physical Const   > C = 2.99792458e+10;  RE = 6.3781366e+8;  GM = 3.986004418e+20;  J2 = 1.0826359e-3 *
* Earth Models     > Gravity gradient includes 1st nonspherical geopotential term (J2)              *
*                  > Magnetic field generated using IGRF2000 spherical harmonic terms up to order 3    *
* Orbit Params     > a = 1227119174.;  e = 0.0044319;  i = 109.84188;  [w = (M+w) - M];              *
*                  > RAAN     = { 109.051226,     0.34255584}                                       *
*                  > M+w Quad = { 319.420422,  2298.97906233, 1.77236871e-7}                        *
*                  > M Quad   = { 107.570967,  2299.19307317, 1.70332257e-7}                        *
* Satellite        > Moments: I1 = 1.2710e+8;  I3 = 1.3140e+8                                       *
*                  > MagTorq: R(eq)= 23.600;  flat= 0.00%;  MCScl= 2.50;  MCShl= 0.00;  sig= 1.0000e+17*
* ICs (angle,rate) > theta = (170.67, 1.0e-016);  phi = (90.36, 1.0e-016);  psi = (0.00, 2.69865)    *
*                  > epoch = -2678.446690 (JD2K)                                                    *
* Units are cgs & degrees; some values implicit/not directly used depending on integration method    *
*****************************************************************************************************
```

Data Run for Figure 4.10

```
*****************************************************************************************************
*                              Lageos Spin Dynamics Model Version 5.0                               *
*                                                                                                   *
*    Date: 14 Dec 2002      Start Time: 15:26:36      Stop Time: 16:44:12      Total Time:  77:35.9    *
* ------------------------------------------------------------------------------------------------- *
* Run Description  > Data Analysis                                                                  *
*                  > Input Source = Program Defaults                                                *
* Integrator Ctrl  > Driver = lageos_spin_rk (order 8(5,3) Runge-Kutta method)                      *
*                  > RTOL = 1.00e-10;  ATOL = 2.22e-16                                              *
*                  > MaxStepSize = 100.0s;   MaxInternalTimeValue = 1.00e+7                         *
* Physical Const   > C = 2.99792458e+10;  RE = 6.3781366e+8;  GM = 3.986004418e+20;  J2 = 1.0826359e-3 *
* Earth Models     > Gravity gradient includes 1st nonspherical geopotential term (J2)              *
*                  > Magnetic field generated using IGRF2000 spherical harmonic terms up to order 3    *
* Orbit Params     > a = 1227119174.;  e = 0.0044319;  i = 109.84188;  [w = (M+w) - M];              *
*                  > RAAN     = { 109.051226,     0.34255584}                                       *
*                  > M+w Quad = { 319.420422,  2298.97906233, 1.77236871e-7}                        *
*                  > M Quad   = { 107.570967,  2299.19307317, 1.70332257e-7}                        *
* Satellite        > Moments: I1 = 1.2798e+8;  I3 = 1.3070e+8                                       *
*                  > MagTrq: OrbFrq=1  R(eq)=23.534  flat=0.00%  MCScl=2.50  MCShl=0.00  sig=1.0000e+17*
* ICs (angle,rate) > theta = (165.70, 1.0e-016);  phi = (3.56, 1.0e-016);  psi = (0.00, 3.0764)     *
*                  > epoch (start) = -2826.333206 JD2K;    stop = 22173.666794 JD2K                 *
* Units are cgs & degrees; some values implicit/not directly used depending on integration method    *
*****************************************************************************************************
```

Appendix B – Lageos Software Package Source Code

The following is a listing of the source code for custom portions of the Lageos spin model (see 3.7.5 for the contents of each module and explanations of the specific files). Source code for external packages integrated into the model can be found at the references provided in the text. Electronic copies of the source code for the model will be provided on demand. Send requests to the author's permanent email address: sewilli@alum.mit.edu; please include a brief statement citing the interest in the problem and the intended purpose for the software.

The Modules

Model Control

LMAIN.C

```
#include    "Lincl.h"
#include    "Lglob.h"

/*****************************************************************************************
PROGRAM:  LAGEOS Spin Dynamics Model
  Models the body spin dynamics of the LAGEOS1 satellite.
INPUTS/OUTPUTS/RETURN VALUE:  none
INCLUDES/EXTERNAL REFERENCES:
  Lincl.h : stdlib.h, stdio.h
  - Lprot.h   : bsstep(), deriv(), deriv_shell_de(), deriv_shell_rk(), dump_headers(),
              : dump_log(), file_ops(), get_params(), global_alloc(), lageos_main(),
              : lageos_optmain(), lageos_spin_de(), lageos_spin_nr(), lageos_spin_rk()
  - Lparams.h : _FOPT
  Lglob.h : fp_log, gf_driver, gf_out
COMMENTS:
   First release written by Warner A. Miller and Dan Holz of the Theoretical Astrophysics
  Group (T-6 MS B288), Theoretical Division, Los Alamos National Laboratory
   Updates written by Scott Williams, a graduate student in Applied Mathematics at North
  Carolina State University.  Details about the model (physical, numerical, software) are
  available in a PhD dissertation "The Lageos Satellite: A Comprehensive Spin Model and
  Analysis" accessed via the NC State Library website.
   I have tried to be thorough and accurate with accompanying comments but there were times
  when the pace of development exceeded the pace of commentary.  Therefore, there may be
  mismatches between commentary and code.
USE:
   I envisioned writing a detailed "User's Guide" to this package but time and other demands
  interfered.  The commetn header (such as this one) with each function should be enough to
  understand the use of the software.  Also refer to the *.h files, (Lparams.h and Lopt.h
  in particular) for more detailed information.  Finally, the aforementioned dissertation
  contains a section devoted to the software development and features of this package.
REUSE:
    This software is intended for the general use of anyone interested in the subject matter.
  I would love to claim that it is completely portable but in fact it most likely is not.  My
  development environment uses the Mingw port (Win 32) of GCC (the GNU Compiler Collection)
  and so I expect the software should work well with most any GCC based compilers.
    Still, there are a couple of things to be aware of.  First, I have utilized several new
  features of the C99 standard and possibly some language extensions which may be peculiar
  to GCC.  These features are enabled with the compiler command -std=gnu9x.  I expect most
  other compilers have similar capability to extend their feature set.
    Second, this package includes Fortran source code as well as C.  The gcc package includes
  compilers for several different languages (C, C++, Fortran, Java, ?) and the generic
  compiler (gcc) automatically selects the appropriate compiler for a given source-code
  file and then seemlessly links the resulting objects.  To enable the C to/from Fortran
  interface, though, it is necessary to add the link commands -lg2c -lm (in that order).  The
  g2c.h header file will also need to be included wherever C code is calling Fortran.
    Finally, extensive use has been made of the GNU Scientific Library (GSL).  GSL is a
  comprehensive library for numerical computing written natively in ANSI C.  It is freely
  available on the web and easy to 'install'.  Once the GSL headers and libraries are in
  locations that your compiler can find them, you will need to issue the link commands
  -lgsl -lgslcblas.  If the compiler supports inline functions, the GSL routines will run
  faster if the compiler flag -DHAVE_INLINE is issues.

  NOTE: The three routines lageos_spin_xx() (_nr, _rk, _de) really ought to be merged into a
        single control routine "lageos_spin" which loops from node to node (much as is already
        done in _rk & _de) with separate evolve_xx() that performs the simpler task of
        integrating from the input node to the output node. Not too difficult to do but a
        diversion from my current efforts . . .
MODIFICATION HISTORY:
  ????  Warner Miller       First Release
        Dan Holz
  9711  Scott Williams      Improved model of earth's magnetic field and restructured code
  0210  Scott Williams      Sig changes in program architecture & data handling; modest
```

```
                           tweaks/improvements to specific model elements & parameters
  0211  Scott Williams     Revised physical model components, inserted additional integration
                           packages, added optimization capability
*****************************************************************************************/
int    main (void)
{
    switch (_FOPT) {
      case  2: lageos_main();                    // do both!
      case  1: lageos_optmain();  break;
      default: lageos_main();
    }

    system("PAUSE");                             /* "Press any key to continue..."*/
    return 0;
}
// proxy for main()
void    lageos_main(void)
{
    int i;

    global_alloc(1);                             // allocate global matrices & vectors
    get_params(0);                               // get pre-defined program parameters
    global_alloc(2);                             // allocate more global matrices & vectors

    i = (gf_out == 2) ? -1 : 0;                  // switch: file stream or null stream
    file_ops(i);                                 // open data output file streams
    dump_log(fp_log, 0, NULL, NULL, NULL, NULL, NULL, NULL);
    dump_headers();
    file_ops(1);

    switch (gf_driver) {
      case 1:  lageos_spin_nr ((void *) deriv, (void *) bdstep); break;
      case 2:  lageos_spin_rk ((void *) deriv_shell_rk); break;
      case 3:  lageos_spin_de (deriv_shell_de); break;
      default: lageos_spin_nr ((void *) deriv, (void *) bsstep);
    }

    dump_log(stdout, 0, NULL, NULL, NULL, NULL, NULL, NULL);
    fflush(fp_log);

    // screen print angular velocity data at end of run (deprecate?)
    for (i=0; i<g_tcnt;i++)
        fprintf(stdout,"g_w[%2d].*  is %8.2f %10.2e %10.2e %10.2e %7.2f %7.2f %7.2f %7.2f "
                "%7.2f %7.2f\n", i, g_w[i].jd2k, g_w[i].mag, g_w[i].ax,
                g_w[i].tr, g_w[i].Blon*M_DPR, g_w[i].Blat*M_DPR, g_w[i].ra*M_DPR,
                g_w[i].dec*M_DPR, g_w[i].Olon*M_DPR, g_w[i].Olat*M_DPR);
    // - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
    global_alloc(0);                             // free dynamically allocated memory
    file_ops(2);                                 // close data output file streams

}

/*****************************************************************************************
PROGRAM: global_alloc
    Allocates/de-allocates dynamic memory for global matrices and arrays
INPUTS/OUTPUTS/RETURN VALUE:
  f_mode : 0=free memory; 1=allocate memory b4 get_params; 2=allocate memory after get_params
INCLUDES/EXTERNAL REFERENCES:
  Lincl.h : malloc.h, gsl_matrix.h, gsl_vector.h, FLAG, euler_data{}, spin_data{}
  - Lprot.h   : lageos_error(),
  - Lparams.h : _NOUT, _NVAR,
  Lglob.h : gT_o2e, gT_e2b, gT_b2L, gT_b2Lr1, gT_b2Lr2, gT_b2Lr3, gv_B, gv_Bb, gv_NmagB,
          : gv_work, g_euler, g_L, g_nout, g_orb, g_out, g_state0, g_w
COMMENTS:
  This routine allocates memory for the default vector/matrix/array/struct sizes hard-coded
  into the program via the Lparams.h header.  If/when the capability is added to read program
  parameters in from a file, some of variables allocated here will need to be reallocated
  within the read-in routines.  Freeing the memory can still be done entirely within the
  context of this routine.
*****************************************************************************************/
void    global_alloc(FLAG f_mode)
{
    int  i;
```

```
    switch (f_mode)
    {
      // allocate memory needed by get_params and/or for general use
      case 1:    {
          gT_o2e    = gsl_matrix_calloc(3,3);
          gT_e2b    = gsl_matrix_calloc(3,3);
          gT_b2L    = gsl_matrix_calloc(3,3);
          gT_b2Lr1  = gsl_matrix_row(gT_b2L, 0);
          gT_b2Lr2  = gsl_matrix_row(gT_b2L, 1);
          gT_b2Lr3  = gsl_matrix_row(gT_b2L, 2);
          g_state0  = gsl_vector_calloc(_NVAR);
          g_out     = gsl_vector_calloc(_NOUT);
          gv_B      = gsl_vector_calloc(3);
          gv_Bb     = gsl_vector_calloc(3);
          gv_Nmagb  = gsl_vector_calloc(3);
          gv_work   = gsl_vector_calloc(3);
          g_orb.v_r = gsl_vector_calloc(3);
          break; }
      // allocate memory needing get_params info
      case 2:    {
          /* allocate storage constructs for targeted outputs if required; space needed is
             1st + last + all targets between <= 2 + g_nout - g_outndx  b/c g_outndx is # of
             elements skipped to get to first target in output range                          */
          i = 2 + g_nout - g_outndx;
          g_euler = (struct euler_data *) malloc (i*(sizeof (struct euler_data)));
            if (g_euler==0)  lageos_error("Couldn't allocate memory for g_euler structure");
          g_L     = (struct spin_data *)  malloc (i*(sizeof (struct spin_data)));
            if (g_L==0)      lageos_error("Couldn't allocate memory for g_L structure");
          g_w     = (struct spin_data *)  malloc (i*(sizeof (struct spin_data)));
            if (g_w==0)      lageos_error("Couldn't allocate memory for g_w structure");
          break; }

      case 0:   {                                  // free allocated memory
          gsl_matrix_free(gT_o2e);
          gsl_matrix_free(gT_e2b);
          gsl_matrix_free(gT_b2L);
          gsl_vector_free(g_state0);
          gsl_vector_free(g_out);
          gsl_vector_free(gv_B);
          gsl_vector_free(gv_Bb);
          gsl_vector_free(gv_Nmagb);
          gsl_vector_free(gv_work);
          gsl_vector_free(g_orb.v_r);

          // de-allocate targeted output storage
          free ((struct euler_data *) g_euler);
          free ((struct spin_data *) g_L);
          free ((struct spin_data *) g_w);
          break; }
    }
}

/*****************************************************************************************
PROGRAM:  intgr8_init
  Common initialization routine for the various lageos_spin_xx() driver routines.
INPUTS/OUTPUTS/RETURN VALUE:
  nvar - dimension of the system
  y    - UNIT OFFSET vector [1 ... nvar] with initial state of dependent variables
  h    - signed initial stepsize (s)
  x    - initial local time (s)
  stop - local stop time (s)
  dsav - signed local time interval (s) for recurring output
  xsav - local time (s) of next recurrent output node
  nout - array index of next targeted output node
  xout - local time (s) of next targeted output node
INCLUDES/EXTERNAL REFERENCES:
  Lincl.h : math.h, gsl_vector.h, _JD2S()
  Lglob.h : g_atol0, g_epoch, g_h0, g_nout, g_out,  g_outndx, g_rtol0, g_save, g_start,
          : g_state0, g_stop, g_tcnt
COMMENTS:
MODIFICATION HISTORY:
  0210  Scott Williams     First Release
*****************************************************************************************/
void    intgr8_init(const int nvar, double *y, double *h, double *x, double *stop,
```

```
            double *dsav, double *xsav, int *nout, double *xout)
{
    int i;
    // retrieve program inputs and initialize variables
    for (i=1; i<=nvar; i++)  y[i] = gsl_vector_get(g_state0, i-1);
    *h = g_h0;                                      // initial step size (signed)
    *x = 0;                                         // set independent variable & output anchor
    *stop   = _JD2S(g_stop - g_start);              // set local stop time (s from g_start)
    g_epoch = g_start;                              // initialize global time tracker (JD2K)
    *dsav   = g_save ? _JD2S(g_save) : *stop;       // signed interval for recurring output
    *xsav   = *dsav;                                // local time of next recurrent output
    *nout   = g_outndx;                             // indx of time of next targeted output
    g_tcnt  = 0;                                    // targeted output counter (incl 1st & last)
    g_rtol  = g_rtol0;
    g_atol  = g_atol0;

    // If targeted output off or no target nodes in integration direction, set xout to stop
    if (!g_nout || *nout == g_nout)  *xout = *stop;
    // Else set xout to 1st target time in direction of integration & convert to local time
    else  {
        *xout = _JD2S(gsl_vector_get(g_out, *nout) - g_epoch);
        *xout = (fabs(*xout) > fabs(*stop)) ? *stop : *xout;  // but no need to go beyond stop
    }
}


/*****************************************************************************************
PROGRAM:  lageos_spin_nr
  Driver routine for the LAGEOS satellite spin dynamics model using numerical integration
  extrapolation method(s) adapted from Numerical Recipes in C, 2nd Ed., Chapter 16.  Advances
  spin state through externally specified time interval (JD2K).  Intermediate outputs can be
  generated and written to output files and/or stored in internal arrays as directed by external
  mode switches.  See Lparams.h for more info on integration control & data output choices.
INPUTS/OUTPUTS/RETURN VALUE:
  derivs - user supplied function that computes the derivatives (3rd argument) of the dependent
           variables (2nd argument) at a specified value of the independent variable (1st
           argument)
INCLUDES/EXTERNAL REFERENCES:
  Lincl.h : stdio.h, math.h, gsl_vector.h, FLAG, _JD2S(), _S2JD()
  - Lprot.h   : bsstep(), dump_data(), fcncntRead(), fcncntReset(), intgr8_init(), lageos_warn()
  - Lparams.h : _HMIN, _NVAR, _TINY
  Lglob.h : fp_log, g_epoch, g_hmax, g_maxstp, g_nout, g_phimod, g_psimod, g_rtol, g_tcnt
COMMENTS:
  - V 1.0 : functionality originally split between main() and odeint().  Initializations and
    memory allocations were don in main().  Odeint() was adapted from Numerical Recipes in C,
    2nd Edition., as a generic driver for integration routines with adaptive stepsize control,
    modified only to include outputs to data files.
  - V 2.0 : removed superficial layer between the old main() & odeint() and taylored the
    generic odeint() to this application.  Added an upper bound on the allowable step size was
    added for consistency with assumptions used to derive the equations of motion; eliminated
    internal storage arrays; added output of integrator performance monitoring.
  - V 2.4 : moved control parameters into global variables in place of hard-wired #defines;
    added feature to keep phi & psi bounded (modulo values to specified range.
  - The integration method is hard-wired into the routine largely because different integration
    packages have different calling formats so code modification needed required for any
    package changes.
MODIFICATION HISTORY:
  9711  Scott Williams     First Release
  0210  Scott Williams     Incremental tweaks & improvements (see comments)
  0210  Scott Williams     Added variable length arrays (VLAs) & targeted data output times
*****************************************************************************************/
void    lageos_spin_nr(void (*derivs)(const double, const double *, double *),
                       void (*step)(double *, const double *, const int, double *, double,
                                    const double, const double *, double *, double *,
                                    void (*derivs)(const double, const double *, double *)))
//void    lageos_spin_nr(void (*derivs)(const double, const double *, double *))
{
    FLAG    f_done=0, f_sing=0;
    char    warnMsg[100];
    int     i=_NVAR+1, nout=0;
    long    nphi=1, npsi=1, nstp=0, nmax=0, nok=0, nbad=0;
    double  x, h, hdid, hnext, temp, dsav, xsav, xout, stop;
    double  y[i], yscal[i], dydx[i];
```

```
    // retrieve program inputs and initialize variables
    intgr8_init(_NVAR, y, &h, &x, &stop, &dsav, &xsav, &nout, &xout);
    fcncntReset();                                  // reset Lnrode derivs call counter

    // output initial state
    dump_data(x, y, nphi, npsi, nstp, nok, nbad, nmax, fcncntRead());
    g_tcnt++;

/* ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ MAIN INTEGRATION LOOP ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
   Advance state from g_start to g_stop, outputting intermediate results
-------------------------------------------------------------------------------------------*/
    for (nstp=1; nstp<=g_maxstp && !f_done; nstp++) {

        // get derivatives of y at x and set scaling vector for error estimation
        (*derivs)(x, y, dydx);
        for (i=1; i<=_NVAR; i++)  yscal[i] = fabs(y[i])+fabs(dydx[i]*h)+_TINY;

        // prevent overshoot of specific output times and/or stop time
        if (fabs(x+h) >= fabs(xout)) {  h = xout - x;  f_done = 1;  }

        // advance y from x to x+hdid and bookkeep the result (achieve h or not)
        (*step)(y, dydx, _NVAR, &x, h, g_rtol, yscal, &hdid, &hnext, derivs);
        if (fabs(hdid) < fabs(h))  { nbad++; f_done = 0; }
        else  nok++;

        // modulo phi & psi to desired interval (multiple of 2pi)
        if (y[2]>g_phimod)  {nphi++; y[2] -= g_phimod;}
          else if (y[2]<0)  {nphi--; y[2] += g_phimod;}
        if (y[3]>g_psimod)  {npsi++; y[3] -= g_psimod;}
          else if (y[3]<0)  {npsi--; y[3] += g_psimod;}

        // test if near theta angle singularities
        f_sing = g_save ? (y[1] < 2.91e-4 || y[1] > (M_PI - 2.91e-4)) : 0;

        // either at targeted output node or the end
        if (f_done) {
            // outputting data now so can reset other output cases
            f_sing = 0;
            xsav   = x + dsav;
            dump_data(x, y, nphi, npsi, nstp, nok, nbad, nmax, nstp + fcncntRead());
            g_tcnt++;
            f_done = !(fabs(x) < fabs(stop));

            // set xout to next output time and continue if not at end of time interval
            if (!f_done) {
                if (nout < g_nout-1) {
                    nout++;
                    xout = _JD2S(gsl_vector_get(g_out, nout) - g_epoch);
                    xout = (fabs(xout) > fabs(stop)) ? stop : xout;
                }
                else  xout = stop;
            }
        }

        // output recurrent data and/or if near theta singularity
        if (f_sing || (fabs(x) >= fabs(xsav))) {
            xsav = x + dsav;
            dump_data(x, y, nphi, npsi, nstp, nok, nbad, nmax, nstp + fcncntRead());
        }

        // check for valid next step size & respond if too big or too small
        temp = fabs(hnext);
        if (temp > fabs(g_hmax))  { nmax++; h = g_hmax; }
        else if (temp <= fabs(_HMIN*x)) {
            sprintf(warnMsg,"Stepsize too small in lageos_spin_nr: h = %12.4e", hnext);
            lageos_warn(fp_log, g_epoch+_S2JD(x), warnMsg);
        }
        else  h = hnext;

        // reset x to zero if too big (smaller relative stepsizes possible if stay near zero)
        if (fabs(x) > g_reset) {
            stop    -= x;
            xsav    -= x;
            xout    -= x;
```

```
            g_epoch += _S2JD(x);
            x = 0;
        }
    }

    // check if routine exceeded stepsize
    if (nstp >= g_maxstp)  lageos_error("Too many steps in routine lageos_spin_nr");

}

/*****************************************************************************************
PROGRAM:  lageos_spin_rk
  Driver routine for the LAGEOS satellite spin dynamics model using the DOP853 Runge-Kutta
  numerical integration method of Hairer & Wanner (see Ldop853.h for more info).  Advances
  spin state through externally specified time interval (JD2K).  Intermediate outputs can be
  generated and written to output files and/or stored in internal arrays as directed by external
  mode switches.  See Lparams.h for more info on integration control & data output choices.
INPUTS/OUTPUTS/RETURN VALUE:
  derivs - user supplied function that computes the derivatives (4th argument) of the dependent
           variables (3rd argument) at a specified value of the independent variable (2nd
           argument); the 1st argument is the dimension of the system
INCLUDES/EXTERNAL REFERENCES:
  Lincl.h : stdio.h, math.h, gsl_vector.h, FLAG, _JD2S(), _S2JD()
  - Lprot.h   : dop853(), dump_data(), intgr8_init(), lageos_warn(), maxcntRead(),
              : naccptRead(), nfcnRead(),nrejctRead(), nstepRead()
  - Lparams.h : _HMIN, _NVAR, _TINY
  Lglob.h : fp_log, gf_idir, g_epoch, g_hmax, g_maxstp, g_nout, g_phimod, g_psimod, g_rtol,
          : g_tcnt
COMMENTS:
   Skeleton similar to lageos_spin_nr() except that the main integration loop advances from
  node to node (e.g., targeted outputs) rather than one basic integration step at a time. This
  makes for better utilization of the internal efficiencies of the integration package.
   The DOP853 integration package itself is used virtually unalterred.  A thorough introduction
  and explanation of the method is provided in the header file Ldop853.h.
MODIFICATION HISTORY:
  0210  Scott Williams     First Release
  0211  Scott Williams     Separated node-to-node portion into evolve_rk() in anticipation of
                           migration to generic lageos_spin() with evolve_xx() (_rk, _de, _nr)
*****************************************************************************************/
void    evolve_rk (FLAG *f_intgr8, int nv, double *x, double *y, double *xnext, double h,
            unsigned long *nstp, unsigned long *nok, unsigned long *nbad, unsigned long *nmax,
            unsigned long *nfcn, void (*derivs)(unsigned, double, double *, double *));
#define  STP_MRGN  10
void    lageos_spin_rk(void (*derivs)(unsigned, double, double *, double *))
{
    FLAG    f_tst, f_intgr8;
    int     i=_NVAR+1, nout=0;
    long    nphi=1, nphi0 = 0, npsi=1, npsi0 = 0;
    unsigned long  nstp=0, nok=0, nbad=0, nfcn=0, nmax=0;
    double  x, absx, h, dsav, xsav, xout, xnext, stop, phisv, psisv;
    double  y[i];

    // retrieve program inputs and initialize variables
    intgr8_init(_NVAR, y, &h, &x, &stop, &dsav, &xsav, &nout, &xout);

    // output initial state
    dump_data(x, y, nphi, npsi, nstp, nok, nbad, nmax, nfcn);
    g_tcnt++;

/* ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ MAIN INTEGRATION LOOP ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
   Advance state from g_start to g_stop, outputting intermediate results; local time variable
   x resets when |x| >= g_reset
-----------------------------------------------------------------------------------------------*/
    for (;;)
    {
        // determine next output node
        if (gf_idir) xnext = GSL_MIN( g_reset, GSL_MIN(xsav, GSL_MIN(xout, stop)));
        else         xnext = GSL_MAX(-g_reset, GSL_MAX(xsav, GSL_MAX(xout, stop)));

        // integrate from x to xnext
        evolve_rk (&f_intgr8, _NVAR+1, &x, y, &xnext, h, &nstp,&nok,&nbad,&nmax,&nfcn, derivs);

        // modulo phi & psi to desired interval (multiple of 2pi)
        phisv = y[2];                           // save original values so can continue
```

```
        psisv = y[3];                               // integration if not also resetting x
        if (fabs(y[2]) > g_phimod)  {
            nphi  = nphi0 + (long) (y[2]/g_phimod);
            y[2]  = fmod(y[2], g_phimod);
            if (y[2] < 0) { nphi--;  y[2] += g_phimod; }
        }
        if (fabs(y[3]) > g_psimod)  {
            npsi  = npsi0 + (long) (y[3]/g_psimod);
            y[3]  = fmod(y[3], g_psimod);
            if (y[3] < 0) { npsi--;  y[3] += g_psimod; }
        }

        // If done, output final results and exit
        absx = fabs(x);
        if (absx >= fabs(stop)) {
            dump_data(x, y, nphi, npsi, nstp, nok, nbad, nmax, nfcn);
            g_tcnt++;                               // # targeted outputs on exit
            g_epoch += _S2JD(x);                    // JD2K on exit
            break;
        }

        // Output targeted/recurrent data and set new nodes
        f_tst = (absx >= fabs(xout));               // Targeted output node
        if (f_tst || (absx >= fabs(xsav))) {
            dump_data(x, y, nphi, npsi, nstp, nok, nbad, nmax, nfcn);
            xsav = x + dsav;                        // reset recurrent output
            if (fabs(xsav - stop) < STP_MRGN)       // inhibit redundent output at end of
                xsav = 2*stop;                      // time interval; force xnext = stop

            if (f_tst) {                            // targeted output node bookkeeping
                g_tcnt++;
                nout++;
                // get local time of next target node
                if (nout < g_nout)  xout = _JD2S(gsl_vector_get(g_out, nout) - g_epoch);
                if (fabs(xout - stop) < STP_MRGN)  // inhibit redundent output at end of
                    xout = 2*stop;                  // time interval; force xnext = stop
            }
        }
        // reset x to zero and set remaining local time variables accordingly
        if (absx >= g_reset) {
            stop    -= x;
            xsav    -= x;
            xout    -= x;
            g_epoch += _S2JD(x);
            x = 0;
            nphi0 = nphi;
            npsi0 = npsi;
        }
        else {
            y[2] = phisv;                           // set back to original values to
            y[3] = psisv;                           // continue integration
        }
    }
}
// see comment in lageos_spin_rk above
void    evolve_rk (FLAG *f_intgr8, int nv, double *x, double *y, double *xnext, double h,
            unsigned long *nstp, unsigned long *nok, unsigned long *nbad, unsigned long *nmax,
            unsigned long *nfcn, void (*derivs)(unsigned, double, double *, double *))
{
    char    warnMsg[100];
    double  habs;

    while ((gf_idir && *x < *xnext) || (!gf_idir && *x > *xnext)) {
        // take an integration step
        *f_intgr8 = dop853 (nv, derivs, *x, y, *xnext, &g_rtol, &g_atol, 0, NULL, 0,
                            stdout, GSL_DBL_EPSILON, 0, 0, 0, 0, g_hmax, h, g_maxstp,
                            1, 0, 0, NULL, 0);

        *x = xRead();                               // gets value of x on output
        h  = hRead();                               // predicted stepsize for next call
        *nstp += nstepRead();                       // total number of steps used
        *nok  += nacceptRead();                     // number of accepted steps
        *nbad += nrejctRead();                      // number of rejected steps
        *nmax += maxcntRead();                      // number of times used max stepsize
```

```
        *nfcn += nfcnRead();                    // number of function calls

        // check for valid next stepsize
        habs = fabs(h);
        if (habs > fabs(g_hmax))  h = g_hmax;
        else if (habs <= fabs(_HMIN*(*x)) || *f_intgr8 == -3) {
            sprintf(warnMsg,"Stepsize too small in lageos_spin_rk: h = %12.4e", h);
            lageos_warn(fp_log, g_epoch+_S2JD(*x), warnMsg);
        }
        // evaluate integrator output flags
        switch (*f_intgr8)
        {
          case -2:   {
              lageos_warn(fp_log, g_epoch+_S2JD(*x), "Step count overflow in dop853; "
                  "resetting x to 0 & retrying step");
              *xnext = *x;
              break; }
          case -3:   {
              lageos_warn(fp_log, g_epoch+_S2JD(*x), "Stepsize underflow in dop853; "
                  "resetting x to 0 & retrying step");
              *xnext = *x;
              break; }
          case -1: lageos_error ("Invalid/inconsistent inputs to dop853()"); break;
          case -4: lageos_error ("ODE may be 'stiff' - dop853() unsuitable for this "
                                 "type of problem"); break;
          default:
        }
    }
}
/*****************************************************************************************
PROGRAM:  lageos_spin_de
  Driver routine for the LAGEOS satellite spin dynamics model using the variable order,
  variable step Adams Bashforth Moulton PECE (Predictor-Estimator-Corrector-Estimator) multi-
  step method of Shampine (see Lshode.f for more info).  Advances spin state through externally
  specified time interval (JD2K).  Intermediate outputs can be generated and written to output
  files and/or stored in internal arrays as directed by external mode switches.  See Lparams.h
  for more info on integration control & data output choices.
INPUTS/OUTPUTS/RETURN VALUE:
  f - user supplied function that computes the derivatives (3rd argument) of the dependent
      variables (2nd argument) at a specified value of the independent variable (1st argument)
INCLUDES/EXTERNAL REFERENCES:
  Lincl.h : stdio.h, math.h, g2c.h, gsl_vector.h, FLAG, _JD2S(), _S2JD()
  - Lprot.h   : ode(), dump_data(), intgr8_init(), lageos_warn()
  - Lparams.h : _NVAR
  Lglob.h : fp_log, g_atol, gf_idir, g_epoch, g_nout, g_phimod, g_psimod, g_rtol, g_tcnt
COMMENTS:
  1) Skeleton nearly identical to lageos_spin_rk().  Major difference is the employment of f2c
     style declarations to ensure compatibility with the integration routine which is fortran
     source code.  The ode subroutine must be linked with the libraries -lg2c -lm
  2)   de_ relies on historical data points and so must be re-initialized if there are any
     changes to the underlying variables.  The routine loses can become both inefficient and
     unreliable if the re-initialization occurs too frequently.
       In this code, phi & psi are modulo'ed for output purposes but then returned to their
     original values before the next iteration of the integrator.  However, when x exceeds the
     reset interval, all the variables (including phi & psi) are reset and so, too, is the
     integrator.
     ***  It is therefore recommended that de_ not be used with small reset interval values  ***
MODIFICATION HISTORY:
  0210  Scott Williams     First Release
*****************************************************************************************/
void    lageos_spin_de(U_fp f(const doublereal *x, const doublereal *y, doublereal*yp))
{
    FLAG        f_tst;
    integer     f_deout=1, neqn=_NVAR, iwork[5], kstp, kfcn;
    char        warnMsg[100];
    int         nout=0;
    long        nphi=1, npsi=1, nphi0 = 0, npsi0 = 0;
    unsigned long  nstp=0, nok=0, nbad=0, nfcn=0, nmax=0;
    double      absx, dsav, xsav, xout, stop, phisv, psisv;
    doublereal  x, xnext;
    doublereal  y[_NVAR+1], work[100+21*(_NVAR+1)];

    // retrieve program inputs and initialize variables
    intgr8_init(_NVAR, y, &xnext, &x, &stop, &dsav, &xsav, &nout, &xout);
```

```
    // output initial state
    dump_data(x, y, nphi, npsi, nstp, nok, nbad, nmax, nfcn);
    g_tcnt++;
/* ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ MAIN INTEGRATION LOOP ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
   Advance state from g_start to g_stop, outputting intermediate results; local time variable
   x resets when |x| >= g_reset
-----------------------------------------------------------------------------------------------*/
    for (;;)
    {
        // determine next output node
        if (gf_idir) xnext = GSL_MIN( g_reset, GSL_MIN(xsav, GSL_MIN(xout, stop)));
        else         xnext = GSL_MAX(-g_reset, GSL_MAX(xsav, GSL_MAX(xout, stop)));

        //while (fabs(x) < fabs(xnext)) {
        while ((gf_idir && x < xnext) || (!gf_idir && x > xnext)) {
            ode((U_fp) f, &neqn, &y[1], &x, &xnext, (doublereal *) &g_rtol,
                (doublereal *) &g_atol, &f_deout, work, iwork, &kstp, &kfcn);
            nstp += kstp;                           // total number of steps used
            nfcn += kfcn;                           // number of function calls

            switch (f_deout)
            {
              // error tolerances too small
              case -3:
              case  3:   {
                  sprintf(warnMsg,"ode increased error tolerances: g_rtol = %.2e & g_atol"
                      " = %.2e", g_rtol, g_atol);
                  lageos_warn(fp_log, g_epoch+_S2JD(x), warnMsg);
                  break; }
              case -5:
              case  5:   {
                  lageos_warn(fp_log, g_epoch+_S2JD(x), "ode reports system may be stiff; "
                              "use alternate method if warning persists");
                  break; }
              case -6:
              case  6:  lageos_error ("Invalid/inconsistent inputs to ode_()"); break;
              default:
            }
        }

        // modulo phi & psi to desired interval (multiple of 2pi)
        phisv = y[2];                               // save original values so can continue
        psisv = y[3];                               // integration if not also resetting x
        if (fabs(y[2]) > g_phimod)  {
            nphi  = nphi0 + (long) (y[2]/g_phimod);
            y[2]  = fmod(y[2], g_phimod);
            if (y[2] < 0) { nphi--;  y[2] += g_phimod; }
        }
        if (fabs(y[3]) > g_psimod)  {
            npsi  = npsi0 + (long) (y[3]/g_psimod);
            y[3]  = fmod(y[3], g_psimod);
            if (y[3] < 0) { npsi--;  y[3] += g_psimod; }
        }

        // If done, output final results and exit
        absx = fabs(x);
        if (absx >= fabs(stop)) {
            dump_data(x, y, nphi, npsi, nstp, nok, nbad, nmax, nfcn);
            g_tcnt++;                               // # targeted outputs on exit
            g_epoch += _S2JD(x);                    // JD2K on exit
            break;
        }

        // Output targeted/recurrent data and set new nodes
        f_tst = (absx >= fabs(xout));               // Targeted output node
        if (f_tst || (absx >= fabs(xsav))) {
            dump_data(x, y, nphi, npsi, nstp, nok, nbad, nmax, nfcn);
            xsav = x + dsav;                        // reset recurrent output
            if (fabs(xsav - stop) < STP_MRGN)       // inhibit redundent output at end of
                xsav = 2*stop;                      // time interval; force xnext = stop

            if (f_tst) {                            // targeted output node bookkeeping
                g_tcnt++;
```

```
                nout++;
                // get local time of next target node
                if (nout < g_nout)  xout = _JD2S(gsl_vector_get(g_out, nout) - g_epoch);
                if (fabs(xout - stop) < STP_MRGN)  // inhibit redundent output at end of
                    xout = 2*stop;                 // time interval; force xnext = stop
            }
        }
        // reset x to zero and set remaining local time variables accordingly
        if (absx >= g_reset) {
            stop    -= x;
            xsav    -= x;
            xout    -= x;
            g_epoch += _S2JD(x);
            x = 0;
            nphi0 = nphi;
            npsi0 = npsi;
            f_deout  = 1;                          // re-initialize ode
        }
        else {
            y[2] = phisv;                          // set back to original values to
            y[3] = psisv;                          // continue integration
        }
    }
}
//*********************************************************************************************
```

LIO.C

```
#include    "Lincl.h"
#include    "Lextern.h"

/*********************************************************************************************
PROGRAM:  banner
  Facilitates repeated character and border printing
INPUTS/OUTPUTS/RETURN VALUE:
  f_nl = endline flag: 0 = don't append '\n'; 1 = append '\n'
  Format: repeat cleft(char) lleft(#) times; pad with lpad(#) white spaces; repeat cmid(char)
  lmid(#) times; pad with rpad(#) white spaces; repeat cright(char) lright(#) times
INCLUDES/EXTERNAL REFERENCES:
  Lincl.h : stdio.h, FLAG
*********************************************************************************************/
void    banner(FILE *fptr, FLAG f_nl, char cleft, char cmid, char cright, int lleft,
            int lpad, int lmid, int rpad, int lright)
{
    int i;
    for (i=0; i<lleft; i++)  fprintf(fptr, "%c",  cleft);
    if (lpad)                fprintf(fptr, "%*c", lpad, ' ');
    for (i=0; i<lmid; i++)   fprintf(fptr, "%c",  cmid);
    if (rpad)                fprintf(fptr, "%*c", rpad, ' ');
    for (i=0; i<lright; i++) fprintf(fptr, "%c",  cright);
    if (f_nl)                fprintf(fptr,"\n");
}

/*********************************************************************************************
PROGRAM:  dump_data
  Prints data to output files and to screen; uses global storage arrays to pre-process data so
  the results may be retained if desired (see Lparams.h targeted output information)
INPUTS/OUTPUTS/RETURN VALUE:
  x      - independent variable (time in seconds)
  y      - unit offset (range [1..._NVAR] vector of dependent variables (Euler angles Theta,
           Phi, and Psi in radians & their respective rates) at time x
  nphi, npsi - number of times modulo of phi (psi) taken
  nstp, nok, nbad, nmax - number of integration steps (total, good, bad, max) where good =
            achieved suggested step; bad = didn't; max = took maximum allowed step (g_hmax)
  nfcn   - number of function calls used
INCLUDES/EXTERNAL REFERENCES:
  Lincl.h   : stdio.h, math.h, gsl_blas.h, gsl_matrix.h, gsl_vector.h, _S2JD(), M_2PI, M_DPR,
  - Lprot.h : dump_log(), euler_rot(), file_ops(), sat_posn(), sincos()
  Lextern.h : fp_euler, fp_angvel, fp_angmom, fp_log, fp_orbit, g_epoch, g_euler, g_I1, g_I3,
            : g_L, g_orb, g_orbp, g_tcnt, g_w
```

```
COMMENTS:
  - Print formats for primary variables chosen to yield 0.1 sec (time), 1 arcsec (angle), and,
    for rates on order of unity or less, at least 1 arcsec/s.  Other value formats vary
  - Euler angles represent the rotational transformation from the inertial frame (ECI) to the
    body frame.  The sequence is: rotate phi about ECI z; rotate theta about resulting
    intermediate x; rotate psi about body z.
  - For any right handed cartesian coordinate system, Longitude (lon) and Latitude (lat) are
    spherical angles with lon the counterclockwise angle in the xy plane from the x-axis and
    lat the 'vertical' angle measured from the xy plane.  Co-Latitude (CoLat) is the 90
    degree compliment of lat (90-lat); it is the vertical angle measured from the z axis
  - Right Ascension (ra) & Declination (dec) are the respective Longitude and Co-Latitude
    referenced SPECIFICALLY to the ECI coordinate system
  - Axial & transverse are the respective polar axis and equatorial plane vector components
MODIFICATION HISTORY:
  9711  Scott Williams     First Release
  0109  Scott Williams     Modified output data & formats
  0210  Scott Williams     Extensive formatting and file content changes
  0210  Scott Williams     Added internal array storage for target data sets and reworked
                           derivations to incorporate GSL constructs
*****************************************************************************************/
void    dump_data(const double x, const double *y, const long nphi, const long npsi,
            const unsigned long nstp, const unsigned long nok, const unsigned long nbad,
            const unsigned long nmax, const unsigned long nfcn)
{
    static unsigned long  num=1;
    double  jd2k, dum1;
    double  sc_th[2], sc_phi[2], sc_psi[2];
    struct euler_data *eptr = &g_euler[g_tcnt];
    struct spin_data  *Lptr = &g_L[g_tcnt], *wptr = &g_w[g_tcnt];

    gsl_matrix * T_b2e = gsl_matrix_alloc(3,3);
    gsl_matrix * T_e2o = gsl_matrix_alloc(3,3);
    gsl_vector * v_wb  = gsl_vector_alloc(3);
    gsl_vector * v_we  = gsl_vector_alloc(3);
    gsl_vector * v_wo  = gsl_vector_alloc(3);
    gsl_vector * v_Lb  = gsl_vector_alloc(3);
    gsl_vector * v_Le  = gsl_vector_alloc(3);
    gsl_vector * v_Lo  = gsl_vector_alloc(3);


    // convert time to JD2K format
    jd2k = g_epoch +  _S2JD(x);

    // Print header lines on first call to routine and runtime/screen banners periodically
    if (fabs(jd2k-g_start) < 1e-10) num = 1;      // reset for iterative calls
    if (num % 300 == 0) dump_log(stdout, 0, NULL, NULL, NULL, NULL, NULL, NULL, NULL);
    if (num % 75 == 0)  dump_log(stdout, 1, NULL, NULL, NULL, NULL, NULL, NULL, NULL);

/* ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ Output Log File Data ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ */
    dump_log(fp_log, 2, &jd2k, &num, &nstp, &nok, &nbad, &nmax, &nfcn);
    num++;


/* ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ Output Euler Angles ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
    columns:  | time | theta | phi | psi | nphi | npsi | thetad | phid | psid |  ra  |  dec  |
    units:    |  JD  | . . .  deg  . . . | . . N/A . . | . . .  deg/s   . . . | . . deg  . . |
    nphi & npsi are modulo interval 'revs'; ra & dec locate the body axis in ECI space
-----------------------------------------------------------------------------------------------*/
    eptr->jd2k = jd2k;
    eptr->th   = y[1];
    eptr->ph   = y[2];
    eptr->nph  = nphi;
    eptr->ps   = y[3];
    eptr->nps  = npsi;
    eptr->thd  = y[4];
    eptr->phd  = y[5];
    eptr->psd  = y[6];
    eptr->ra   = fmod(y[2],M_2PI)-M_PI_2;
    eptr->dec  = M_PI_2 - y[1];
    fprintf(fp_euler, "%13.6f ||%11.4f |%11.4f |%11.4f ||%10ld |%10ld ||%14.4e |%14.4e "
            "|%14.4e ||%10.3f |%10.3f\n", eptr->jd2k, eptr->th*M_DPR, eptr->ph*M_DPR,
            eptr->ps*M_DPR, eptr->nph, eptr->nps, eptr->thd*M_DPR, eptr->phd*M_DPR,
            eptr->psd*M_DPR, eptr->ra*M_DPR, eptr->dec*M_DPR);
```

```
/* ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ Output Orbit Parameters ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
    columns:  |  time  | radius |  RAAN  |   Mw   | Mw revs |   M   |   E   |   w   |
    units:    |   JD   |   km   | . . . deg . . . |    #    | .  .  .  deg  .  .  . |
    w & M omitted b/c not included in current orbit model; Mw is net angular position M+w
-------------------------------------------------------------------------------------------------*/
    // compute orbit parameters
    sat_postn(jd2k, &g_orb);
    fprintf(fp_orbit, "%13.6f ||%11.3f |%11.4f ||%11.4f |%9ld ||%9.3f |%9.3f |%9.3f\n",
            jd2k, 1.0e-5*g_orb.r, g_orb.W*M_DPR, g_orb.Mw*M_DPR, g_orb.rev, g_orb.M*M_DPR,
            g_orb.E*M_DPR, g_orb.w*M_DPR);


/* ~~~~~~~~~~~~~~~~ Output Body Frame Angular Velocity ("Spin Vector") Parameters ~~~~~~~~~~~~~~~~
    columns: |  time  |   w   | axial/w | trans/w |  lon  |  lat  | + . . .
    units:   |   JD   | deg/s |  (dimensionless)  | . .  deg  . . |
    w is the scalar magnitude; axial/trans are normalized body frame components along/normal to
    satellite body axis; lon/lat locate the spin vector in the body frame
-------------------------------------------------------------------------------------------------*/
    // compute sine & cosine of Euler Angles & body to ECI transformation matrix
    euler_rot(0, y[1], y[2], y[3], sc_th, sc_phi, sc_psi, 4, T_b2e, NULL);

    // compute body frame angular velocity ("spin vector") and angular momentum
    dum1  = y[5]*sc_th[0];
    gsl_vector_set (v_wb, 0, dum1*sc_psi[0] + y[4]*sc_psi[1]);
    gsl_vector_set (v_wb, 1, dum1*sc_psi[1] - y[4]*sc_psi[0]);
    gsl_vector_set (v_wb, 2, y[5]*sc_th[1] + y[6]);
    wptr->jd2k = jd2k;
    wptr->Blon = atan2(gsl_vector_get(v_wb, 1), gsl_vector_get(v_wb, 0));
    Lptr->Blon = wptr->Blon;                        // body angular momentum has same longitude
    wptr->mag  = gsl_blas_dnrm2(v_wb);
    wptr->ax   = gsl_vector_get(v_wb, 2)/(wptr->mag);
    wptr->Blat = asin(wptr->ax);
    wptr->tr   = cos(wptr->Blat);
    fprintf(fp_angvel, "%13.6f ||%16.6e |%16.6e |%16.6e ||%11.4f |%11.4f |",
            jd2k, wptr->mag*M_DPR, wptr->ax, wptr->tr, wptr->Blon*M_DPR, wptr->Blat*M_DPR);


/* ~~~~ Output Right Ascension/Declination & Orbit Frame Longitude/Latitude of spin vector ~~~~~
    columns: + . . .|  ra (deg)  | dec (deg) |  lon (deg)  | lat (deg) |
-------------------------------------------------------------------------------------------------*/
    // transform spin vector to eci frame
    gsl_blas_dgemv (CblasTrans, 1.0, T_b2e, v_wb, 0.0, v_we);
    wptr->ra  = atan2(gsl_vector_get(v_we, 1), gsl_vector_get(v_we, 0));
    wptr->dec = asin(gsl_vector_get(v_we, 2)/(wptr->mag));

    // compute transformation matrix from ECI to orbit centered frame
    euler_rot(3, g_orbp.i, g_orb.W, g_orb.w, g_orb.sci, g_orb.scW, g_orb.scw, 14, T_e2o, NULL);

    // transform spin vector to orbit frame
    gsl_blas_dgemv (CblasNoTrans, 1.0, T_e2o, v_we, 0.0, v_wo);
    wptr->Olon  = atan2(gsl_vector_get(v_wo, 1), gsl_vector_get(v_wo, 0));
    wptr->Olat  = asin(gsl_vector_get(v_wo, 2)/(wptr->mag));
    fprintf(fp_angvel, "|%10.3f |%10.3f ||%10.3f |%10.3f\n", wptr->ra*M_DPR, wptr->dec*M_DPR,
                       wptr->Olon*M_DPR, wptr->Olat*M_DPR);


/* ~~~~~~~~~~~~~~~~~~~~~~~ Output Body Frame Angular Momentum Parameters ~~~~~~~~~~~~~~~~~~~~~~~
    columns: |  time  |      L     | axial/L | trans/L |  lon  |  lat  | + . . .
    units:   |   JD   | (g cm^2)/s |  (dimensionless)  | . .  deg  . . |
    parameters identically analagous to those for angular velocity
-------------------------------------------------------------------------------------------------*/
    // compute body frame angular momentum
    gsl_vector_set (v_Lb, 0, g_I1*gsl_vector_get(v_wb, 0));
    gsl_vector_set (v_Lb, 1, g_I1*gsl_vector_get(v_wb, 1));
    gsl_vector_set (v_Lb, 2, g_I3*gsl_vector_get(v_wb, 2));
    Lptr->mag  = gsl_blas_dnrm2(v_Lb);
    Lptr->ax   = gsl_vector_get(v_Lb, 2)/(Lptr->mag);
    Lptr->Blat = asin(Lptr->ax);
    Lptr->tr   = cos(Lptr->Blat);
    fprintf(fp_angmom, "%13.6f ||%16.6e |%16.6e |%16.6e ||%11.4f |%11.4f |",
            jd2k, Lptr->mag, Lptr->ax, Lptr->tr, Lptr->Blon*M_DPR, Lptr->Blat*M_DPR);
```

```
/* ~~~~~~~~~~~~~~~ Output RA/Dec & Orbit Frame Lon/Lat of angular momentum vector ~~~~~~~~~~~~~~~~
    columns: + . . .|  ra (deg)  | dec (deg) |  lon (deg)  | lat (deg) |
-------------------------------------------------------------------------------------------*/
    // transform angular momentum vector to eci frame
    gsl_blas_dgemv (CblasTrans, 1.0, T_b2e, v_Lb, 0.0, v_Le);
    Lptr->ra  = atan2(gsl_vector_get(v_Le, 1), gsl_vector_get(v_Le, 0));
    Lptr->dec = asin(gsl_vector_get(v_Le, 2)/(Lptr->mag));

    // transform spin vector to orbit frame
    gsl_blas_dgemv (CblasNoTrans, 1.0, T_e2o, v_Le, 0.0, v_Lo);
    Lptr->Olon  = atan2(gsl_vector_get(v_Lo, 1), gsl_vector_get(v_Lo, 0));
    Lptr->Olat  = asin(gsl_vector_get(v_Lo, 2)/(Lptr->mag));
    fprintf(fp_angmom, "|%10.3f |%10.3f ||%10.3f |%10.3f\n", Lptr->ra*M_DPR, Lptr->dec*M_DPR,
            Lptr->Olon*M_DPR, Lptr->Olat*M_DPR);

    gsl_matrix_free (T_b2e);
    gsl_matrix_free (T_e2o);
    gsl_vector_free (v_wb);
    gsl_vector_free (v_we);
    gsl_vector_free (v_wo);
    gsl_vector_free (v_Lb);
    gsl_vector_free (v_Le);
    gsl_vector_free (v_Lo);

    file_ops(1);  // flush file pointers
}

/*************************************************************************************************
PROGRAM:  dump_headers
  Writes header lines to output files and screen
INPUTS/OUTPUTS/RETURN VALUE: none
INCLUDES/EXTERNAL REFERENCES:
  Lincl.h   : stdio.h
  - Lprot.h : banner(), dump_log()
  Lextern.h : fp_euler, fp_angvel, fp_angmom, fp_log, fp_orbit
*************************************************************************************************/
void    dump_headers(void)
{
    int nc;

    dump_log(fp_log, 1, NULL, NULL, NULL, NULL, NULL, NULL, NULL);
    fprintf(fp_euler,"      time    ||    theta   |     phi    |     psi    ||  phi mod# |"
            "  psi mod# ||     thetad    |      phid     |      psid     ||    RA     |"
            "    Dec%n\n      JD2K    ||     deg    |     deg    |     deg    ||           |"
            "           ||      deg/s    |      deg/s    |      deg/s    ||    deg    |"
            "    deg\n", &nc);
    banner(fp_euler, 1, '=', 0, 0, nc+4, 0, 0, 0, 0);
    fprintf(fp_angvel,"      time    ||     |AngVel|    |      axial      |"
            "    transverse   ||  Body Long |  Body Lat  ||    RA     |    Dec    ||"
            "  OCI Long |  OCI Lat%n\n      JD2K    ||       deg/s     |   (normalized)  |"
            "   (normalized)  ||     deg    |     deg    ||    deg    |    deg    ||"
            "    deg    |    deg\n", &nc);
    banner(fp_angvel, 1, '=', 0, 0, nc+2, 0, 0, 0, 0);
    fprintf(fp_angmom,"      time    ||     |AngMom|    |      axial      |"
            "    transverse   ||  Body Long |  Body Lat  ||    RA     |    Dec    ||"
            "  OCI Long |  OCI Lat%n\n      JD2K    ||    (g cm^2)/s   |   (normalized)  |"
            "   (normalized)  ||     deg    |     deg    ||    deg    |    deg    ||"
            "    deg    |    deg\n", &nc);
    banner(fp_angmom, 1, '=', 0, 0, nc+2, 0, 0, 0, 0);
    fprintf(fp_orbit,"      time    ||   radius   |    RAAN    ||    M + w   | M+w Rev# ||"
            "     M    |     E    |     w   %n\n      JD2K    ||     km     |     deg    ||"
            "     deg    |          ||    deg   |    deg   |    deg   \n", &nc);
    banner(fp_orbit, 1, '=', 0, 0, nc+1, 0, 0, 0, 0);
}

/*************************************************************************************************
PROGRAM:  dump_log
  Writes program status data and data run information to the log output file and to the screen
INPUTS/OUTPUTS/RETURN VALUE:
  fptr   - pointer to output stream
  f_mode - mode flag: 0 = Runtime header, 1 = column header, 2 = data line
  jd2k   - time (JD2K)
  num    - data line number
  nstp, nok, nbad, nmax, nfcn - see dump_data() comments
```

```
INCLUDES/EXTERNAL REFERENCES:
  Lincl.h   : stdio.h, stdlib.h, string.h, time.h, gsl_vector.h, FLAG, M_DPR
  - Lprot.h : banner(), elapsed_time(), expcat()
  Lextern.h : gf_driver, gf_grav, gf_mag, gf_mfreq, g_atol, g_c, g_clatm, g_desc,
            : g_gm, g_hmax, g_I1, g_I3, g_insrc, g_J2, g_lonm, g_magscl, g_magshl,
            : g_mdm, g_oblcore, g_orbp, g_re, g_reset, g_rcore, g_rtol,
            : g_sigcore, g_start, g_state0, g_title
COMMENTS:
  - All outputs are directed to stdout (screen) in addition to the output file designated by
    fptr.  For screen output only, call with stdout as the file pointer argument
  - Automatically updates timekeeping line in output file on each data line call
MODIFICATION HISTORY:
  0210  Scott Williams     First Release
*****************************************************************************************/
void    dump_log(FILE *fptr, const FLAG f_mode, const double *jd2k, const unsigned long *num,
            const unsigned long *nstp, const unsigned long *nok, const unsigned long *nbad,
            const unsigned long *nmax, const unsigned long *nfcn)
{
    FILE          *fout;
    time_t        now;
    static FLAG   f_1st=1;
    static char   *nowstr, start[30], *drvstr;
    static int    len, padl, padr, width = 105;
    static long   cltime, savepos, currpos;
    static double s;

    fout  = fptr;
    time(&now);
    nowstr = ctime(&now);
    if (f_1st) {
        f_1st = 0;                                  // do this stuff once
        strcpy(start,nowstr);                       // save start time
        len  = strlen(g_title);                     // compute variable format lengths
        padr = (width - len)/2;
        padl = (2*padr+len==width) ? padr : padr+1;
    }

    switch (f_mode) {
//********************************* PRINT RUNTIME HEADER ************************************
      case 0:  {
        int    nc, nc2, ex1, ex2, ex3, ex4;
        double bs1, bs2, bs3, bs4;
        if (!f_1st)  banner(stdout, 1, '=', 0, 0, width, 0, 0, 0, 0);
        for (;;) {
            // Print program title & wersion number
            banner(fout, 1, '*', 0, 0, width, 0, 0, 0, 0);
            fprintf(fout, "%-*c%s%*c\n", padr, '*', g_title, padl, '*');
            banner(fout, 1, '*', 0, '*', 1, width-2, 0, 0, 1);

            // Print date/time/duration line and save pointer to it's location
            if (fout != stdout) savepos = ftell(fout);
            fprintf(fout,"*    Date: %.2s %.3s %.4s      Start Time: %.8s      Stop Time: %.8s"
                         "      Total Time: %3ld:%04.1f     *\n", start+8, start+4, start+20,
                         start+11, nowstr+11, cltime, s);
            banner(fout, 1, '*', '-', '*', 1, 1, width-4, 1, 1);

            // Print run description info
            fprintf(fout,"* Run Description  %n> ",  &nc2);
            if (*g_desc) {
                fprintf(fout,"%s%n", g_desc, &nc);
                fprintf(fout,"%*s\n%-*s> ", width-(nc+nc2+2), "*", nc2, "*");
            }
            fprintf(fout,"Input Source = %s%n", g_insrc, &nc);
            fprintf(fout,"%*s\n", width-(nc+nc2+2), "*");

            // Print integrator control parameters
           switch (gf_driver) {
              case 1:  drvstr = "_nr (Bader-Deuflhard extrapolation method)"; break;
              case 2:  drvstr = "_rk (order 8(5,3) Runge-Kutta method)";      break;
              case 3:  drvstr = "_de (variable order/variable step Adams PECE method"; break;
              default: drvstr = "_nr (Bulirsch-Stoer extrapolation method)";  break;
            }
            fprintf(fout,"* Integrator Ctrl  > Driver = lageos_spin%s%n", drvstr, &nc);
            fprintf(fout,"%*s\n", width-nc, "*");
```

```
bs1 = expcat(g_rtol, &ex1);
bs2 = expcat(g_atol, &ex2);
fprintf(fout,"%-*s> RTOL = %.2fe%+d;  ATOL = %.2fe%+d%n", nc2, "*",
            bs1, ex1, bs2, ex2, &nc);
fprintf(fout,"%*s\n", width-nc, "*");
bs1 = expcat(g_reset, &ex1);
fprintf(fout,"%-*s> MaxStepSize = %.1fs;   MaxInternalTimeValue =%5.2fe%+d%n",
            nc2, "*", g_hmax, bs1, ex1, &nc);
fprintf(fout,"%*s\n", width-nc, "*");

// Print physical constant values
bs1 = expcat(g_c, &ex1);
bs2 = expcat(g_re, &ex2);
bs3 = expcat(g_gm, &ex3);
bs4 = expcat(g_J2, &ex4);
fprintf(fout,"* Physical Const   > C = %.8fe%+d;  RE = %.7fe%+d;  GM = %.9fe%+d;  "
            "J2 = %.7fe%+d%n", bs1, ex1, bs2, ex2, bs3, ex3, bs4, ex4, &nc);
fprintf(fout,"%*s\n", width-nc, "*");

// Print earth model parameters
if (!gf_grav)
    fprintf(fout,"* Earth Models     > Gravity gradient does not include "
                "nonspherical geopotential terms%n", &nc);
else
    fprintf(fout,"* Earth Models     > Gravity gradient includes 1st nonspherical "
                "geopotential term (J2)%n", &nc);
fprintf(fout,"%*s\n", width-nc, "*");
if (gf_mag == 0) {
    fprintf(fout,"%-*s> Magnetic field generated using static dipole with "
                "following parameters:%n", nc2, "*", &nc);
    fprintf(fout,"%*s\n", width-nc, "*");
    fprintf(fout,"%-*s> Dipole Moment = %.7e; Location = %.4f lon, %.4f co-lat%n",
                nc2, "*", g_mdm,  M_DPR*g_lonm, M_DPR*g_clatm, &nc);
}
else if (gf_mag <= 10) {
    fprintf(fout,"%-*s> Magnetic field generated using IGRF2000 spherical "
                "harmonic terms up to order %d%n", nc2, "*", gf_mag, &nc);
} else {
    fprintf(fout,"%-*s> Magnetic field generated using static dipole fixed at "
                "IGRF2000 %4d epoch location%n", nc2, "*", gf_mag, &nc);
}
fprintf(fout,"%*s\n", width-nc, "*");

// Print Lageos orbit determination parameters
fprintf(fout,"* Orbit Params     > a = %#.0f;  e = %.7f;  i = %.5f;  [w = (M+w) "
            "- M];%n", g_orbp.a, g_orbp.e, M_DPR*g_orbp.i, &nc);
fprintf(fout,"%*s\n", width-nc, "*");
fprintf(fout,"%-*s> RAAN     = {%11.6f, %14.8f}%n", nc2,"*",M_DPR*g_orbp.W[0],
            M_DPR*g_orbp.W[1], &nc);
fprintf(fout,"%*s\n", width-nc, "*");
bs1 = expcat(M_DPR*g_orbp.Mw[2], &ex1);
fprintf(fout,"%-*s> M+w Quad = {%11.6f, %14.8f, %.8fe%+d}%n", nc2, "*",
            M_DPR*g_orbp.Mw[0], M_DPR*g_orbp.Mw[1], bs1, ex1, &nc);
fprintf(fout,"%*s\n", width-nc, "*");
if (g_orbp.f_sin) {
    fprintf(fout,"%-*s> M+w Sin  = {%11.6f, %14.8f, %13.8f}%n", nc2, "*",
            M_DPR*g_orbp.Mw[3], M_2PI/g_orbp.Mw[4], g_orbp.Mw[5], &nc);
    fprintf(fout,"%*s\n", width-nc, "*");
}
bs1 = expcat(M_DPR*g_orbp.M[2], &ex1);
fprintf(fout,"%-*s> M Quad   = {%11.6f, %14.8f, %.8fe%+d}%n", nc2, "*",
            M_DPR*g_orbp.M[0], M_DPR*g_orbp.M[1], bs1, ex1, &nc);
fprintf(fout,"%*s\n", width-nc, "*");


// Print Lageos satellite model parameters
bs1 = expcat(g_I1, &ex1);
bs2 = expcat(g_I3, &ex2);
fprintf(fout,"* Satellite        > Moments: I1 = %.4fe%+d;  I3 = %.4fe%+d%n",
            bs1, ex1, bs2, ex2, &nc);
fprintf(fout,"%*s\n", width-nc, "*");
bs1 = expcat(g_sigcore, &ex1);
fprintf(fout,"%-*s> MagTrq: OrbFrq=%d  R(eq)=%.3f  flat=%.2f%%  MCScl=%.2f  "
            "MCShl=%.2f  sig=%.4fe%+d%n", nc2, "*", gf_mfreq, g_rcore,
```

```
                    100*(1-g_oblcore), g_magscl, g_magshl-1.0, bs1, ex1, &nc);
            fprintf(fout,"%*s\n", width-nc, "*");

            // Print initial spin state and corresponding epoch
            fprintf(fout,"* ICs (angle,rate) > theta = (%.2f, %#.2g);  phi = (%.2f, %#.2g);"
                    "  psi = (%.2f, %.6g)%n", M_DPR*gsl_vector_get(g_state0,0),
                    M_DPR*gsl_vector_get(g_state0,3), M_DPR*gsl_vector_get(g_state0,1),
                    M_DPR*gsl_vector_get(g_state0,4), M_DPR*gsl_vector_get(g_state0,2),
                    M_DPR*gsl_vector_get(g_state0,5), &nc);
            fprintf(fout,"%*s\n", width-nc, "*");
            fprintf(fout,"%-*s> epoch (start) = %.6f JD2K;    stop = %.6f JD2K%n",
                    nc2, "*", g_start, g_stop, &nc);
            fprintf(fout,"%*s\n", width-nc, "*");

            fprintf(fout,"* Units are cgs & degrees; some values implicit/not directly used "
                    "depending on integration method%n",&nc);
            fprintf(fout,"%*s\n", width-nc, "*");
            banner(fout, 1, '*', 0, 0, width, 0, 0, 0, 0);

            // Done if screen print accomplished
            if (fout == stdout) break;
            fout = stdout;
        }
        break; }

//********************************** PRINT COLUMN HEADER ************************************
      case 1:  {
        for (;;) {
            banner(fout, 1, '=', 0, 0, width, 0, 0, 0, 0);
            fprintf(fout," runtime || sat time || data ||integratn |  delta  || success  "
                    "|  retry  | maxstep || Function |  delta   \n");
            fprintf(fout," mm:ss.s ||   JD2K   || line#||  steps   |  intstp ||  steps   "
                    "|  steps  |  steps  ||  calls   |   Fcn    \n");
            banner(fout, 1, '=', 0, 0, width, 0, 0, 0, 0);
            if (fout == stdout) break;
            fout = stdout;
        }
        break; }

//********************************** PRINT DATA ELEMENTS ************************************
      default: {
        static unsigned long  oldstp, oldfcn, dstp, dfcn;
        if (fabs(*jd2k-g_start) < 1e-10) oldstp = oldfcn = 0;  // reset for iterative calls
        dstp   = (*nstp)-oldstp;
        dfcn   = (*nfcn)-oldfcn;
        elapsed_time(&cltime, &s);
        for (;;) {
            fprintf(fout, "%3ld:%04.1f ||%9.2f ||%5lu ||%9lu |%8ld ||%9lu |%8lu |%8lu "
                   "||%9lu |%8lu\n", cltime, s, *jd2k, *num, *nstp, dstp, *nok, *nbad,
                   *nmax, *nfcn, dfcn);
            if (fout == stdout) break;
            // Replace time line in output file with updated time info
            currpos = ftell(fout);
            fseek(fout, savepos, SEEK_SET);
            fprintf(fout,"*    Date: %.2s %.3s %.4s      Start Time: %.8s      Stop Time: %.8s"
                    "      Total Time: %3ld:%04.1f     *\n", start+8, start+4, start+20,
                    start+11, nowstr+11, cltime, s);
            fseek(fout, currpos, SEEK_SET);
            fout = stdout;
        }

        oldstp = *nstp;
        oldfcn = *nfcn;
      }
    }
}

/*****************************************************************************************
PROGRAM:  file_ops
  Performs output file maintenance operations (open, close, flush)
INPUTS/OUTPUTS/RETURN VALUE:
  f_mode - mode flag: 0 = open files for standard output,
                     -1 = open temporary file streams for faux output
                      1 = flush output streams,
```

```
                    2 = close files
INCLUDES/EXTERNAL REFERENCES:
  Lincl.h   : stdio.h, FLAG
  - Lprot.h : lageos_error()
  Lextern.h : fp_euler, fp_angvel, fp_angmom, fp_log, fp_orbit
*********************************************************************************************/
void    file_ops(const FLAG f_mode)
{
    switch (f_mode)
    {
        case -1:   {    // no file output so just set pointers to NULL stream
            fp_log    = stdout;
            if ((fp_euler  = fopen("NUL:","w")) == NULL)
                lageos_error("Could not open temporary euler angle output file");
            if ((fp_angvel  = fopen("NUL:","w")) == NULL)
                lageos_error("Could not open temporary euler angle output file");
            if ((fp_angmom  = fopen("NUL:","w")) == NULL)
                lageos_error("Could not open temporary euler angle output file");
            if ((fp_orbit  = fopen("NUL:","w")) == NULL)
                lageos_error("Could not open temporary euler angle output file");
        /* the above is theoretically better b/c it doesn't even create a tmp file, but here's
           an approach using the tmpfile() function just for reference:
            if ((fp_euler  = tmpfile()) == NULL)
                lageos_error("Could not open temporary euler angle output file");             */
            break; }
        case 0:    {    // open files for output
            if ((fp_euler  = fopen("l_euler.txt","w")) == NULL)
                lageos_error("Could not open euler angle output file");
            if ((fp_angvel = fopen("l_angvel.txt","w")) == NULL)
                lageos_error("Could not open angular velocity output file");
            if ((fp_angmom = fopen("l_angmom.txt","w")) == NULL)
                lageos_error("Could not open angular momentum output file");
            if ((fp_log    = fopen("l_log.txt","w")) == NULL)
                lageos_error("Could not open program log output file");
            if ((fp_orbit  = fopen("l_orbit.txt","w")) == NULL)
                lageos_error("Could not open orbit output file");
            break; }
        case 1:    {    // flush buffers
            fflush(fp_euler);
            fflush(fp_angvel);
            fflush(fp_angmom);
            fflush(fp_log);
            fflush(fp_orbit);
            break; }
        case 2:    {    // close output files
            fclose (fp_euler);
            fclose (fp_angvel);
            fclose (fp_angmom);
            if (fp_log != stdout) fclose (fp_log);
            fclose (fp_orbit);
            break; }
    }
}

/*********************************************************************************************
PROGRAM: get_params
  Retrieves and initializes program parameters
INPUTS/OUTPUTS/RETURN VALUE:
  f_init - 0 = get (reset) initial values & compute derived parameter values
           1 = only (re)compute derived parameter values (do not reset initial values)
INCLUDES/EXTERNAL REFERENCES:
  Lincl.h     : math.h, gsl_math.h, gsl_sort_vector.h, gsl_vector.h,
              : FLAG, M_2PI, M_RPD, M_3_8PI, SPJD
  - Lprot.h   : lageos_error(),. sincos()
  - Lparams.h : _NOUT, _NVAR, _TINY, A, ATOL, C1, CLATM, DESCR, DRIVER,
              : ECC, ETAA, ETA0, ETAD, ETADD, ETAM, ETAP, ETAT,
              : FETASIN, FGRAV, FLATC, FMAG, FMFREQ, FOUT, GM, H1, HMAX,
              : I1, I3, INC, INSRC, J2, LONM, MAXSTP, M0, MCSCL, MCSHL,
              : MD, MDD, MDM, OUT1, OUT2, OUT3, OUT4, OUT5, OUT6,
              : PHI, PHID, PHIMOD, PSI, PSID, PSIMOD,
              : RCORE, RAAN0, RAAND, RE, RESET, RTOL, SAVE, SIGC,
              : START, STOP, TUAG, THETA, THETAD, TITLE
  Lextern.h   : gf_driver, gf_grav, gf_idir, gf_mag, gf_mfreq, gf_out,
              : g_alph0, g_atol, g_atol0, g_beta1, g_c, g_clatm, g_desc,
```

```
              : g_gm, g_h0, g_hmax, g_I1, g_I3, g_I3dI1, g_Ifact, g_insrc,
              : g_J2, g_J2re2, g_lonm, g_magscl, g_magfac, g_magco1, g_magco2,
              : g_magco3, g_magshl, g_maxstp, g_mdm, g_nout, g_oblcore, g_out,
              : g_outndx, g_orb, g_orbp, g_orbw, g_phimod, g_psimod, g_rcore,
              : g_re, g_reset, g_rtol, g_rtol0, g_save, g_SCclatm, g_sigcore,
              : g_start, g_state0, g_stop, g_taug, g_title, g_vcore
MODIFICATION HISTORY:
  9711  Scott Williams     First Release
  0210  Scott Williams     Incorporated global variables
  0211  Scott Williams     Added new model parameters
*****************************************************************************************/
void    get_params(const FLAG f_init)
{
    int    i;
    double out[_NOUT] = {OUT1, OUT2, OUT3, OUT4, OUT5, OUT6};
    double spin[_NVAR] = {THETA, PHI, PSI, THETAD, PHID, PSID};
    double *vptr;

    if (!f_init) {
        // retrieve program title and version
        g_title  = TITLE;
        g_desc   = DESCR;
        g_insrc  = INSRC;

        // retrieve integration control parameters
        g_rtol0   = RTOL;
        g_atol0   = gf_driver==1 ? RTOL*_TINY : ATOL;
        g_rtol    = g_rtol0;
        g_atol    = g_atol0;
        g_start   = START;
        g_stop    = STOP;
        g_h0      = (gf_idir = (g_stop > g_start)) ? fabs(H1) : -fabs(H1);
        g_maxstp  = MAXSTP;
        g_hmax    = gf_idir ? fabs((double) HMAX) : -fabs((double) HMAX);

        // retrieve variable conditioning parameters
        g_reset  = fabs(RESET);
        g_phimod = fabs(PHIMOD)*M_2PI;
        g_psimod = fabs(PSIMOD)*M_2PI;

        // retrieve data output parameters
        g_save   = gf_idir ? fabs((double) SAVE) : -fabs((double) SAVE);
        gf_out   = FOUT;
        g_nout   = gf_out ? _NOUT : 0;
        for(i=0; i<g_nout; i++)  gsl_vector_set (g_out, i, out[i]);
        gsl_sort_vector(g_out);
        if (gf_idir) {      // find first element > g_start; g_outndx = g_nout if all <= g_start
            vptr = gsl_vector_ptr(g_out, 0);
            for (g_outndx=0; ((g_outndx < g_nout) && ((*vptr) <= g_start)); g_outndx++)
                vptr += g_out->stride;
        }
        else  {             // find first element < g_start; g_outndx = g_nout if all >= g_start
            gsl_vector_reverse (g_out);
            vptr = gsl_vector_ptr(g_out, 0);
            for (g_outndx=0; ((g_outndx < g_nout) && (*vptr >= g_start)); g_outndx++)
                vptr += g_out->stride;
        }

        // retrieve model component option switches
        gf_driver    = DRIVER;
        g_orbp.f_sin = FETASIN;
        gf_grav      = FGRAV;
        gf_mag       = FMAG;
        gf_mfreq     = !(FMFREQ==0);
        if (gf_mag < 0 || gf_mag > 2100 || (gf_mag > 10 && gf_mag < 1900))
            lageos_error ("Invalid mode flag for Earth Magnetic Model");


        // retrieve physical & earth related parameters/constants
        g_c       = C1;
        g_gm      = GM;
        g_J2      = J2;
        g_re      = RE;
        g_mdm     = MDM;
```

```
        g_lonm    = LONM*M_RPD;
        g_clatm   = CLATM*M_RPD;

        // retrieve orbit parameters
        g_orbp.a     = A;
        g_orbp.e     = ECC;
        g_orbp.i     = INC*M_RPD;
        g_orbp.W[0]  = RAAN0*M_RPD;
        g_orbp.W[1]  = RAAND*M_RPD;
        g_orbp.Mw[0] = (ETA0 + (g_orbp.f_sin==1)*ETAM)*M_RPD;
        g_orbp.Mw[1] = ETAD*M_RPD;
        g_orbp.Mw[2] = ETADD*M_RPD;
        g_orbp.Mw[3] = ETAA*M_RPD;
        g_orbp.Mw[4] = M_2PI/ETAT;
        g_orbp.Mw[5] = ETAP;
        g_orbp.M[0]  = M0*M_RPD;
        g_orbp.M[1]  = MD*M_RPD;
        g_orbp.M[2]  = MDD*M_RPD;
        g_orbw       = 2.0*g_orbp.Mw[1]/SPJD;

        // retrieve satellite parameters
        g_rcore   = RCORE;
        g_sigcore = SIGC;
        g_oblcore = 1.0-FLATC;
        g_magscl  = MCSCL;
        g_magshl  = 1.0+MCSHL;
        g_I1      = I1;
        g_I3      = I3;
        g_taug    = TAUG;

        // retrieve initial satellite spin state
        for (i=0; i<_NVAR; i++)  gsl_vector_set (g_state0, i, spin[i]*M_RPD);
    }

    // Compute derived parameters
    g_I3dI1  = g_I3/g_I1;
    g_Ifact  = 1.0 - g_I3dI1;
    g_vcore  = 4*M_PI*gsl_pow_3(g_rcore)/3;
    if (g_magscl != -999)  g_magfac = (g_magscl < 0.0) ? 0.0 : g_magscl;
    g_magshl = (g_magshl < 1.0) ? 1.0 : g_magshl;
    g_magco1 = 2*g_rcore*sqrt(M_2PI*g_sigcore)/g_c;
    g_magco2 = 3*M_3_8PI/g_magco1;
    g_magco3 = -6*M_3_8PI/gsl_pow_2(g_magco1);
    g_beta1  = 3*g_gm*g_taug*(g_I3-g_I1);
    g_J2re2  = 0.5*g_J2*gsl_pow_2(g_re);
    g_alph0  = fmod(M_PI_2+g_lonm, M_2PI);
    sincos(g_clatm, g_SCclatm);

    sincos(g_orbp.i, g_orb.sci);
}

/*****************************************************************************************************
PROGRAM: lageos_error
  For a Lageos Spin Model run-time error, screen prints error message and exits routine
INPUTS/OUTPUTS/RETURN VALUE:
  *error_text - Error message to print (string)
INCLUDES/EXTERNAL REFERENCES:
  Lincl.h : stdlib.h, stdio.h
*****************************************************************************************************/
void    lageos_error(char *error_text)
{
    fprintf(stderr,"Lageos Spin Model run-time error...\n");
    fprintf(stderr,"%s\n",error_text);
    fprintf(stderr,"...now exiting to system...\n");
    system("PAUSE");
    exit(1);
}

/*****************************************************************************************************
PROGRAM: lageos_warn
  Prints a warning message to the standard output stream as well as the stream specified on
  input if different than stdout
INPUTS/OUTPUTS/RETURN VALUE:
  *fout      - pointer to output stream
```

```
  *warn_text - Error message to print (string)
  sim_time   - Simulation time of warning trigger
INCLUDES/EXTERNAL REFERENCES:
  Lincl.h : stdlib.h, stdio.h
*******************************************************************************************/
void    lageos_warn(FILE *fout, double sim_time, char *warn_text)
{
    for (;;) {
        fprintf(fout,"WARNING at %9.2f ||", sim_time);
        fprintf(fout,"%s\n", warn_text);
        if (fout == stdout) break;
        fout = stdout;
    }
}
//*******************************************************************************************
```

LINCL.H

```
// ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~  Macros  ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
#define FLAG     int
#define M_2PI    6.2831853071795864769       // 2*pi                3->2...528676656
#define M_DPR    57.295779513082320768       // degrees per radian  8->7...981548141
#define M_RPD    1.7453292519943295769e-02   // radians per degree  2->2...369076849
#define M_3_8PI  0.11936620731892150183      // 3/8PI               3->2...666282253
#define SPJD     86400.0                     // seconds per Julian Day (exact)
#define _S2JD(a) ((a)/(SPJD))                // convert seconds to Julian Days
#define _JD2S(a) ((a)*(SPJD))                // convert Julian Days to seconds
//-------------------------------------------------------------------------------------------

// ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~  Standard & GCC Header Files  ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
#include    <limits.h>              // Constants for sizes of integer types
#include    <malloc.h>              // Memory management
#include    <math.h>                // Mathematical functions and macros
#include    <stdio.h>               // Functions controlling input/output
#include    <stdlib.h>              // Facilities for number conv., storage alloc.
#include    <string.h>              // Facilities for manipulating/comparing strgs
#include    <time.h>                // Types and funcs for manipulating date/time
#include    <g2c.h>                 // GCC version of f2c
//-------------------------------------------------------------------------------------------

// ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~  GNU Scientific Library Header Files  ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
#include    <gsl/gsl_blas.h>        // Basic linear algebra routines
#include    <gsl/gsl_math.h>        // General math routines & related constants
#include    <gsl/gsl_matrix.h>      // Matrix routines & structures
#include    <gsl/gsl_poly.h>        // Polynomial routines
#include    <gsl/gsl_sort_vector.h> // Vector sort routines
#include    <gsl/gsl_vector.h>      // Vector routines & structures

/* GSL is a collection of routines for numerical computing written in strict ANSI C.  It is
   distributed under the GNU General Public License which basically says it's free to use and
   distribute for non-proprietary purposes.  Website: http://sources.redhat.com/gsl/

   The present GSL package is from the GNUWin32 Project which provides a Win32 port of GNU
   tools.  Website: http://gnuwin32.sourceforge.net/
------------------------------------------------------------------------------------------- */

// ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~  Structures  ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
struct euler_data         // structure for euler angle data sets
{
  double  jd2k;           // JD2K of data set
  double  th;             // euler angle theta (nutation angle)
  double  ph;             // euler angle phi (precession angle)
  long    nph;            // modulo counter for phi
  double  ps;             // euler angle psi (spin angle)
  long    nps;            // module counter for psi
  double  thd;            // time derivative of theta
  double  phd;            // time derivative of phi
  double  psd;            // time derivative of psi
  double  ra;             // right ascension of body axis
  double  dec;            // declination of body axis
```

```
};
struct   orb_params       // structure of parameters for orbit propagation equations
{
  double a;               // semi-major axis (cm) : constant
  double e;               // eccentricity         : constant
  double i;               // inclination (rad)    : constant
  double W[2];            // RAAN (rad)           : linear W[0] + W[1]*t (JD2K)
  double M[3];            // mean anomaly (rad)   : quadratic M[0] + M[1]*t + M[2]*t^2
  double Mw[6];           // net angular position : quadratic Mw[0] + Mw[1]*t + Mw[2]*t^2
                          //      M + w (rad)     : + sinusoidal Mw[3]*sin((t+Mw[5])*Mw[4])
                          //                      : where Mw[4] = 2pi/period
  int    f_sin;           // Sin model flag: 1=use; 0=don't use
  // some orbit parameters are not directly computed (e.g., w) so are omitted from this list
};
struct orb_posn           // structure of current orbit position variables
{
  // note: a, e, & i are constant in current orbit model and stored in the orb_params struct
  double  jd2k;           // JD2K time of satellite position
  double  w;              // argument of perigee (rad)
  double  W;              // right ascension of the ascending node (rad)
  double  M;              // mean anomaly (rad)
  double  E;              // eccentric anomaly E - esin(E) = M; approximated as E = M + e sin M
  double  Mw;             // M + w (rad)
  long    rev;            // revolution # wrt line of nodes
  double  r;              // scalar radius of satellite position (cm)
  gsl_vector * v_r;       // ECI radial position UNIT vector
  double  sci[2];         // sin & cos of inclination
  double  scw[2];         // sin & cos of argument of perigee
  double  scW[2];         // sin & cos of RAAN
  double  scMw[2];        // sin & cos of M+w
};
struct spin_data          // structure for angular velocity and angular momentum data sets
{
  double  jd2k;           // JD2K of data set
  double  mag;            // magnitude of spin or momentum vector
  double  ax;             // axial component of spin/momentum unit vector
  double  tr;             // transverse component of spin/momentum unit vector
  double  Blon;           // Body frame longitude of spin/momentum vector
  double  Blat;           // Body frame latitude of spin/momentum vector
  double  ra;             // right ascension of spin/momentum vector
  double  dec;            // declination of spin/momentum vector
  double  Olon;           // Orbit frame longitude of spin/momentum vector
  double  Olat;           // Orbit frame latitude of spin/momentum vector
};
//-------------------------------------------------------------------------------------------------

// ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~  Fortran Interfacing  ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
#define bspcar   bspcar_
#define igrf     igrf_
#define ode      ode_
//-------------------------------------------------------------------------------------------------

// ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~  Custom Header Files  ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
#include    "Lprot.h"
#include    "Lparams.h"
#include    "Ldop853.h"
//-------------------------------------------------------------------------------------------------
```

LPROT.H

```
/* ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~  Lcontrol folder  ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
   Files: Lmain.c, Lio.c
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ */
// - - - - - - - - - - - - - - - - - - - -   Lmain.c   - - - - - - - - - - - - - - - - - - - - -
void    global_alloc(FLAG f_mode);
void    intgr8_init(const int nvar, double *y, double *h, double *x, double *stop,
            double *dsav, double *xsav, int *nout, double *xout);
void    lageos_main(void);
void    lageos_spin_nr(void (*derivs)(const double, const double *, double *),
                       void (*step)(double *, const double *, const int, double *, double,
```

```
                                    const double, const double *, double *, double *,
                                    void (*derivs)(const double, const double *, double *)));
void    lageos_spin_rk(void (*derivs)(unsigned, double, double *, double *));
void    lageos_spin_de(U_fp f(const doublereal *x, const doublereal *y, doublereal*yp));

// - - - - - - - - - - - - - - - - - - - - - - -  Lio.c  - - - - - - - - - - - - - - - - - - - - - - -
void    banner(FILE *fptr, FLAG f_nl, char cleft, char cmid, char cright, int lleft,
            int lpad, int lmid, int rpad, int lright);
void    dump_data(const double x, const double *y, const long nphi, const long npsi,
            const unsigned long nstp, const unsigned long nok, const unsigned long nbad,
            const unsigned long nmax, const unsigned long nfcn);
void    dump_headers(void);
void    dump_log(FILE *fptr, const FLAG f_mode, const double *jd2k, const unsigned long *num,
            const unsigned long *nstp, const unsigned long *nok, const unsigned long *nbad,
            const unsigned long *nmax, const unsigned long *nfcn);
void    file_ops(const FLAG f_mode);
void    get_params(const FLAG f_mode);
void    lageos_error(char *error_text);
void    lageos_warn(FILE *fout, double sim_time, char *warn_text);

/* ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~  Lnumeric folder  ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
   Files: Ldop853.c, Lnr_c.c, Lshode.f
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ */
// - - - - - - - - - - - - - - - - - - - -  Ldop853.c  - - - - - - - - - - - - - - - - - - - -
// prototypes in header Ldop853.h

//  - - - - - - - - - - - - - - - - - - - -  Lnr_c.c  - - - - - - - - - - - - - - - - - - - -
void    bsstep(double *y, const double *dydx, const int nv, double *xx, double h,
            const double eps, const double *yscal, double *hdid, double *hnext,
            void (*derivs)(const double, const double *, double *));
void    bdstep(double *y, const double *dydx, const int nv, double *xx, double htry,
            const double eps, const double *yscal, double *hdid, double *hnext,
            void (*derivs)(const double, const double *, double *));
float   dfridr(float (*func)(float), float x, float h, float *err, FLAG *f_status, float rtol);
void    mmid(const double *y, const double *dydx, const int nvar, const double xs,
            const double htot, const int nstep, double *yout,
            void (*derivs)(const double, const double *, double *));
void    rzextr(const int iest, double xest, const double *yest, double *yz, double *dy,
            const int nv, double *x, const int nc, double d[nv+1][nc]);
unsigned long fcncntRead(void);
void    fcncntReset(void);

//  - - - - - - - - - - - - - - - - - - - -  Lshode.f  - - - - - - - - - - - - - - - - - - - -
void    ode(U_fp f, integer *neqn, doublereal *y, doublereal *t, doublereal *tout,
            doublereal *relerr, doublereal *abserr, integer *iflag, doublereal *work,
            integer *iwork, integer *nostep, integer *nfcn);
// remaining subroutines are only referenced internally so no need to prototype

/* ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~  Lmodel folder  ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
   Files: Lderivs.c, Ligrf2000.c, Ltools.c
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ */
//  - - - - - - - - - - - - - - - - - - - -  igrf2000.f  - - - - - - - - - - - - - - - - - - - -
void    igrf(integer *iy, integer *nm, doublereal *r__, doublereal *t, doublereal *f,
            doublereal *br, doublereal *bt, doublereal *bf);
void    bspcar(doublereal *teta, doublereal *phi, doublereal *br, doublereal *btet,
            doublereal *bphi, doublereal *bx, doublereal *by, doublereal *bz);

// - - - - - - - - - - - - - - - - - - - - - - -  Ltools.c  - - - - - - - - - - - - - - - - - - - - - - -
void    elapsed_time(long *eminutes, double *eseconds);
void    euler_rot(const FLAG f_mode, const double th, const double phi, const double psi,
            double *sc_th, double *sc_phi, double *sc_psi, const FLAG f_tr,
            gsl_matrix * T_tr, gsl_vector * v_tr);
double  expcat(double x, int *expn);
double  jd2k_2_gha(const double jd2k, const FLAG f_gh0, double *gh0);
void    sincos(double x, double *sc_x);
void    vector_cross(const gsl_vector * a, const gsl_vector * b, gsl_vector * result);

// - - - - - - - - - - - - - - - - - - - - - - -  Lderivs.c  - - - - - - - - - - - - - - - - - - - - - - -
int     deriv(const double x, const double *y, double *dydx);
void    deriv_shell_rk (unsigned nvar, double x, double *y, double *f);
U_fp    deriv_shell_de (const doublereal *x, const doublereal *y, doublereal *yp);
void    eom_free (const double *y, double *dyfree, const double *sc_th, double *work);
void    grav_torques (const gsl_vector * v_r, const double *rsqr, const double *rcube,
            const gsl_matrix * T_e2b, gsl_vector * v_dcosBr, double *Ngrav);
```

```
void    mag_torques(const double *jd2k, const double *rcube, const double *y,
            const double *sc_psi, const double *phidsth, const double *phidcth, double *Nmag);
void    sat_postn(const double jd2k, struct orb_posn * orb);

/* ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~  Loptimize folder  ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
   Files: Lopt.c
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ */
// - - - - - - - - - - - - - - - - - - - - - - Lopt.c - - - - - - - - - - - - - - - - - - - - - - -
void    lageos_optmain(void);
void    dump_params (FILE *fpout, char *subname, const gsl_vector *v_mp, FLAG f_err,
            double *err, FLAG f_endl);
void    dfopt (const gsl_vector * v_mp, void * fpar, gsl_vector * v_fgrad);
void    fdfopt (const gsl_vector * v_mp, void * fpar, double *fcn, gsl_vector * v_fgrad);
double  fopt (const gsl_vector * v_mp, void * fpar);
double  fopt_shell (const double p, void * fpar);
float   fopt_shell_float (const float p);
void    opt_data_init (void);
void    opt_file_ops(const FLAG f_mode);
void    opt_params(void);
// ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

LGLOB.H

```
// program title and version number
char    *g_title;        // program title
char    *g_desc;         // run description
char    *g_insrc;        // input source

// integration control parameters
FLAG     gf_idir;        // integration direction flag 1=increasing time; 0=decreasing time
double   g_atol0;        // absolute error tolerance
double   g_h0;           // signed initial step size
double   g_hmax;         // signed maximum integration stepsize allowed
double   g_maxstp;       // maximum number of integration steps allowed
double   g_rtol0;        // relative error tolerance
double   g_start;        // JD2K integration start time
double   g_stop;         // JD2K integration stop time

// variable conditioning parameters
double   g_phimod;       // upperbound (mult of 2pi) for modulo or Euler angle phi
double   g_psimod;       // upperbound (mult of 2pi) for modulo or Euler angle psi
double   g_reset;        // max value (s) of local independent variable b4 reset to zero

// data output parameters
FILE       *fp_angmom;   // file pointer for angular momentum output file
FILE       *fp_angvel;   // file pointer for angular velocity output file
FILE       *fp_euler;    // file pointer for euler angle output file
FILE       *fp_log;      // file pointer for integration log output file
FILE       *fp_orbit;    // file pointer for orbit position output file
FLAG        gf_out;      // targeted output flag 0=none, 1/2=do with/without file writes
int         g_nout;      // targeted output time array size
gsl_vector *g_out;       // array of targeted output times
long        g_outndx;    // index of closest g_out element to g_start in direction of g_stop
double      g_save;      // signed time interval for writing recurrent output data

// model implementation switches
FLAG     gf_driver;      // driver routine selection flag
FLAG     gf_grav;        // Use 1st zonal term (J2) in gravity model: 1=yes; 0=no
                         // Note: orbit model sin correction flag part of g_orbp struct
FLAG     gf_mag;         // Order of spherical harmonics to use in IGRF magnetic field model
FLAG     gf_mfreq;       // Specifies additive use of orbit frequency in mag torque

// physical & earth related parameters
double   g_alph0;        // earth-fixed longitude of z cross m
double   g_c;            // speed of light
double   g_clatm;        // co-latitude of m
double   g_gm;           // product of gravitational constant & earth mass
double   g_J2;           // 1st order Earth geopotential "zonal' coefficient
double   g_J2re2;        // multiplication factor: 0.5*J2*Re
double   g_mdm;          // earth magnetic dipole moment
```

```
double   g_lonm;         // earth-fixed longitude of m (magnetic dipole vector)
double   g_re;           // equatorial radius of the earth
double   g_SCclatm[2];   // sine & cosine of co-latitude of m

// orbit parameters (for orbit propagation equations)
struct orb_params  g_orbp = {0,0,0,{0,0},{0,0,0},{0,0,0,0,0,0},0};
double             g_orbw;  // 2 x orbital frequency (rad/s)

// satellite paramters
double   g_beta1;        // multiplication factor for computing gravitational torque
double   g_I1;           // Lageos transverse principle moment of inertia
double   g_I3;           // Lageos axial principle moment of inertia
double   g_I3dI1;        // Ratio I3/I1
double   g_Ifact;        // 1.0 - I3/I1
double   g_magscl;       // scaling factor for cylinder adjustment to core mag coeff
double   g_magshl;       // spherical shell correction factor
double   g_magfac;       // magnetization coefficients field orientation-based scaling factor
double   g_magco1;       // derived multiplication factor for computing magnetization coeff
double   g_magco2;       // derived multiplication factor for computing magnetization coeff
double   g_magco3;       // derived multiplication factor for computing magnetization coeff
double   g_oblcore;      // oblate core correction factor for magnetic model = 1-FLATC = Rp/Re
double   g_rcore;        // radius of reference metallic sphere for satellite core
double   g_sigcore;      // effective conductivity of reference metallic sphere for sat. core
double   g_taug;         // gravity torque scaling parameter
double   g_vcore;        // volume of reference metallic sphere for satellite core

// initial satellite spin state (at g_start)
gsl_vector * g_state0;   // initial spin state

// 'targeted' output data set variables (angles in radians)
int                g_tcnt;       // array index placeholder (pts to 'active' array element)
struct euler_data  *g_euler;     // struct array for euler angle data sets
struct spin_data   *g_L;         // struct array for angular momentum data sets
struct spin_data   *g_w;         // struct array for angular velocity data sets

// working variables
double             g_atol;       // absolute error tolerance
double             g_epoch;      // model time (JD2K)
double             g_rtol;       // relative error tolerance
struct orb_posn    g_orb;        // working variable for orbit position computation
gsl_matrix        *gT_o2e;       // Orbit frame to ECI transformation matrix
gsl_matrix        *gT_e2b;       // ECI to body frame transformation matrix
gsl_matrix        *gT_b2L;       // Body to Landau-Lifshitz framer transformation matrix
gsl_vector_view    gT_b2Lr1;     // 1st row of gT_b2L
gsl_vector_view    gT_b2Lr2;     // 2nd row of gT_b2L
gsl_vector_view    gT_b2Lr3;     // 3rd row of gT_b2L
gsl_vector        *gv_B;         // ECI magnetic field vector
gsl_vector        *gv_Bb;        // Body frame magnetic field vector
gsl_vector        *gv_Nmagb;     // Body frame magnetic torque vector
gsl_vector        *gv_work;      // Scratch vector for computations
//*************************************************************************************************
```

LEXTERN.H

```
// program title and version number
extern char    *g_title;
extern char    *g_desc;
extern char    *g_insrc;

// integration control parameters
extern FLAG     gf_idir;
extern double   g_atol0;
extern double   g_h0;
extern double   g_hmax;
extern double   g_maxstp;
extern double   g_rtol0;
extern double   g_start;
extern double   g_stop;

// variable conditioning parameters
```

```
extern double   g_phimod;
extern double   g_psimod;
extern double   g_reset;

// data output parameters
extern FILE       *fp_angmom;
extern FILE       *fp_angvel;
extern FILE       *fp_euler;
extern FILE       *fp_log;
extern FILE       *fp_orbit;
extern FLAG        gf_out;
extern int         g_nout;
extern gsl_vector *g_out;
extern long        g_outndx;
extern double      g_save;

// model implementation switches
extern FLAG     gf_driver;
extern FLAG     gf_grav;
extern FLAG     gf_mag;
extern FLAG     gf_mfreq;

// physical & earth related parameters
extern double   g_alph0;
extern double   g_c;
extern double   g_clatm;
extern double   g_J2;
extern double   g_J2re2;
extern double   g_gm;
extern double   g_mdm;
extern double   g_lonm;
extern double   g_re;
extern double   g_SCclatm[2];

// orbit parameters (for orbit propagation equations)
extern struct orb_params  g_orbp;
extern double             g_orbw;

// satellite paramters
extern double   g_beta1;
extern double   g_I1;
extern double   g_I3;
extern double   g_I3dI1;
extern double   g_Ifact;
extern double   g_magscl;
extern double   g_magsh1;
extern double   g_magfac;
extern double   g_magco1;
extern double   g_magco2;
extern double   g_magco3;
extern double   g_oblcore;
extern double   g_rcore;
extern double   g_sigcore;
extern double   g_taug;
extern double   g_vcore;

// initial satellite spin state (at g_start)
extern gsl_vector * g_state0;
// 'targeted' output data set variables (angles in radians)
extern int                g_tcnt;
extern struct euler_data  *g_euler;
extern struct spin_data   *g_L;
extern struct spin_data   *g_w;

// working variables
extern double              g_atol;
extern double              g_epoch;
extern double              g_rtol;
extern struct orb_posn     g_orb;
extern gsl_matrix         *gT_o2e;
extern gsl_matrix         *gT_e2b;
extern gsl_matrix         *gT_b2L;
extern gsl_vector_view     gT_b2Lr1;
extern gsl_vector_view     gT_b2Lr2;
```

```
extern gsl_vector_view     gT_b2Lr3;
extern gsl_vector         *gv_B;
extern gsl_vector         *gv_Bb;
extern gsl_vector         *gv_Nmagb;
extern gsl_vector         *gv_work;
//*********************************************************************************************
```

Physical Model

LIGRF2000.F – External Package; see [i].

LDERIVS.C

```
#include "Lincl.h"
#include    "Lextern.h"

/*********************************************************************************************
PROGRAM:  deriv, deriv_shell(s)
  Computes the right hand side of the differential equations (Euler angles equations of motion)
  for the spin components of the LAGEOS satellite (Euler Angles).  The shell(s) are simply
  alternate calling formats for consistency with different ode solvers.
INPUTS/OUTPUTS/RETURN VALUE:
  x      - value of the independent variable
  y      - Euler angle state vector (ie, dependent variables) at time = x
  dydx   - resulting state vector derivatives at time = x
  RETURN - GSL_SUCCESS
INCLUDES/EXTERNAL REFERENCES:
  Lincl.h : math.h, gsl_math.h, _S2JD, g2c.h (for shell_de)
  - Lprot.h   : eom_free(), euler_rot(), grav_torques(), mag_torques(), sat_posn()
  - Lparams.h : _TINY
  Lextern.h   : gT_e2b, gv_work, g_epoch, g_I1, g_I3, g_orb
COMMENTS:
   Initial routine (deriv1) implemented equations developed in Miller, Holz, et al paper
  'Spin Dynamics of the LAGEOS Satellite in Support of a Measurement of the Earth's
  Gravitomagnetism'.  The routine was later updated to improve computational efficiency and
  a better magnetic model (accounting for the tilt of the magnetic axis) was inserted.
   The present effort has streamlined this routine by breaking the equations into essential
  modules.  In order of reference, they are:
  - satellite orbital position
  - free motion equations
  - gravitational torques
  - magnetic torques
MODIFICATION HISTORY:
    ????  Warner Miller     First Release
          Dan Holz
    9710  Scott Williams    Optimized for efficiency
    9711  Scott Williams    Added tilt between earth spin & mag axes
    0210  Scott Williams    Broke routine into modules for essential components (see comments)
*********************************************************************************************/
void    deriv_shell_rk (unsigned nvar, double x, double *y, double *f)
{
    deriv(x, y, f);
}
//---------------------------------------------------------------------------------------------
U_fp    deriv_shell_de (const doublereal *x, const doublereal *y, doublereal *yp)
{
    const double *py   = y-1;
    double *dydx = yp-1;

    deriv (*x, py, dydx);
    return 0;
}
//---------------------------------------------------------------------------------------------
int     deriv(const double x, const double *y, double *dydx)
```

```
{
    double  rsqr, rcube, jd2k, work1;
    double  sc_th[2], sc_phi[2], sc_psi[2], Ngrav[3], Nmag[3], work[5];

    // already have derivatives of first three variables
    dydx[1] = y[4];    dydx[2] = y[5];    dydx[3] = y[6];

    // convert time to JD2K format
    jd2k = g_epoch + _S2JD(x);

    // determine position of the satellite at time x
    sat_postn(jd2k, &g_orb);
    rsqr  = gsl_pow_2(g_orb.r);
    rcube = rsqr*g_orb.r;

    // compute sine & cosine of Euler Angles & ECI to body transformation matrix
    euler_rot(0, y[1], y[2], y[3], sc_th, sc_phi, sc_psi, 4, gT_e2b, NULL);
    if (fabs(sc_th[0])<_TINY)  sc_th[0] = GSL_SIGN(sc_th[0])*_TINY;

// - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
    // Compute 'free' components of the Euler Equations
    eom_free (y, &dydx[4], sc_th, work);

// - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
    // compute body frame components of gravitational torques
    grav_torques (g_orb.v_r, &rsqr, &rcube, gT_e2b, gv_work, Ngrav);

// - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
    // compute body frame components of magnetic torques
    mag_torques(&jd2k, &rcube, y, sc_psi, &work[0], &work[2], Nmag);

// - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
    // compute forced components of Euler Equations
    Nmag[0] += Ngrav[0];
    Nmag[1] += Ngrav[1];
    //Nmag[2] += Ngrav[2];   ==> Ngrav[2] = 0!

    dydx[4]  += (Nmag[0]*sc_psi[1] - Nmag[1]*sc_psi[0])/g_I1;
    work1     = (Nmag[1]*sc_psi[1] + Nmag[0]*sc_psi[0])/g_I1;
    work1     = work1/sc_th[0];
    dydx[5]  += work1;
    dydx[6]  += Nmag[2]/g_I3 - work1*sc_th[1];

    return GSL_SUCCESS;
}

/*****************************************************************************************************
PROGRAM:  eom_free
  Computes the free motion components of the euler angles equations of motion and stores s
INPUTS/OUTPUTS/RETURN VALUE:
  y      - UNIT OFFSET euler angle state vector
  sc_th  - array containing pre-computed sin & cos of theta
  dyfree - ZERO OFFSET array with right hand sides for y[4] through y[6]
  work   - 5 element array to store computed products
INCLUDES/EXTERNAL REFERENCES:
  Lincl.h   : math.h
  Lextern.h : g_I3dI1, g_Ifact
COMMENTS:
  - Assumes sc_th[0] has already been preconditioned to avoid a divide by zero issue
*****************************************************************************************************/
void    eom_free (const double *y, double *dyfree, const double *sc_th, double *work)
{
    work[4] = y[4]/sc_th[0];
    work[3] = g_I3dI1 * y[6];
    work[2] = y[5] * sc_th[1];
    work[1] = g_Ifact*work[2] - work[3];
    work[0] = y[5] * sc_th[0];

    dyfree[0] =  work[0] * work[1];
    dyfree[1] = -work[4] * (work[1] + work[2]);
    dyfree[2] =  work[4] * (sc_th[1]*work[1] + y[5]);
}
```

```
/*************************************************************************************************
PROGRAM:  grav_torques
  Computes the body frame components of the gravitational torque on the Lageos satellite
INPUTS/OUTPUTS/RETURN VALUE:
  v_r      - ECI satellite position unit vector (gsl_vector type)
  rsqr     - pointer to computed square of satellite's radius (i.e., magnitude of v_r squared)
  rcube    - pointer to computed cube of the satellite's radius
  T_e2b    - transformation matrix FROM ECI to Body frame (gsl_matrix type)
  v_dcosBr - gsl type vector to store computed direction cosines between the body axes and -v_r
  Ngrav    - resulting vector (std 3 dim array) of body axis satellite torques
INCLUDES/EXTERNAL REFERENCES:
  Lincl.h   : math.h, gsl_blas.h, gsl_matrix.h, gsl_vector.h
  Lextern.h : g_beta1, g_J2re2
COMMENTS:
  - The ECI representations of the satellite body axes are merely the rows of the ECI to Body
    transformation matrix
MODIFICATION HISTORY:
    0210  Scott Williams    Separated this module from deriv() to run as a subroutine
    0211  Scott Williams    Added J2 zonal term for higher accuracy calculations
*************************************************************************************************/
void    grav_torques (const gsl_vector * v_r, const double *rsqr, const double *rcube,
            const gsl_matrix * T_e2b, gsl_vector * v_dcosBr, double *Ngrav)
{
    double   beta, beta1, c13, c23, c33;

// - - - - - - - - - - - - - - - - - Spherical Earth Gravity Torque - - - - - - - - - - - - - - -
    // compute direction cosines of body axes with unit radial vector TOWARD earth
    gsl_blas_dgemv (CblasNoTrans, -1.0, T_e2b, v_r, 0.0, v_dcosBr);
    beta   = g_beta1/(*rcube);
    c13    = gsl_vector_get (v_dcosBr, 0);
    c23    = gsl_vector_get (v_dcosBr, 1);
    c33    = gsl_vector_get (v_dcosBr, 2);
    beta1  = beta*c33;

    Ngrav[0] =   beta1 * c23;
    Ngrav[1] = - beta1 * c13;
    //Ngrav[2] =   0;              // =0 b/c of axial symmetry: I1=I2

// - - - - - - - - - - - - - - - - - Oblate Earth Gravity Torque - - - - - - - - - - - - - - - -
    if (gf_grav) {
        double   beta2, a1, a2, a3, ce13, ce23, ce33, r3;

        beta2 = g_J2re2*beta/(*rsqr);

        // sin of sat latitude = r3 b/c already unitized
        r3    = gsl_vector_get (v_r, 2);

        // direction cosines of body axes with ECI z--just 3rd column of e2b matrix
        ce13  = gsl_matrix_get (T_e2b, 0, 2);
        ce23  = gsl_matrix_get (T_e2b, 1, 2);
        ce33  = gsl_matrix_get (T_e2b, 2, 2);

        // multiplicative coefficients for J2 torque correciton
        a1    = 5*c33*(1 - 7*gsl_pow_2(r3));       // 5*c33*(1 - 7sin(lat))
        a2    = - 10*r3;
        a3    = - 2*ce33;

        Ngrav[0] += beta2*(a1*c23 + a2*(c23*ce33 + c33*ce23) + a3*ce23);
        Ngrav[1] -= beta2*(a1*c13 + a2*(c13*ce33 + c33*ce13) + a3*ce13);
        //Ngrav[2] += 0;                            // =0 b/c of axial symmetry: I1=I2
    }
}

/*************************************************************************************************
PROGRAM:  mag_torques
  Computes the body frame components of the magnetic torque on the Lageos satellite
INPUTS/OUTPUTS/RETURN VALUE:
  jd2k    - pointer to JD2K value of independent variable
  rcube   - pointer to computed cube of the satellite's radius (i.e., magnitude of v_r cubed)
  y       - Euler angle state vector at time = jd2k
  sc_psi  - array containing sin (sc_psi[0]) & cosine ([1]) of euler angle psi (y[3])
  phidsth - pointer to computed value of y[5]*sin(y[1])
  phidcth - pointer to computed value of y[5]*cos(y[1])
```

```
  Nmag    - resulting vector (std 3 dim array) of body axis satellite torques
INCLUDES/EXTERNAL REFERENCES:
  Lincl.h   : math.h, gsl_blas.h, gsl_math.h, gsl_matrix.h, gsl_vector.h, M_2PI, M_3_8PI
  - Lprot.h : bspcar(), euler_rot(), igrf(), jd2k_2_gha(), vector_cross
  Lextern.h : gf_mag, gf_mfreq, gT_b2L, gT_b2Lr1, gT_b2Lr2, gT_b2Lr3, gT_e2b, gv_B, gv_Bb,
            : gv_Nmagb, gv_work, g_alph0, g_clatm, g_magco1, g_magco2, g_magco3, g_magscl
            : g_magshl, g_mdm, g_oblcore, g_orb, g_orbw, g_SCclatm, g_vcore
COMMENTS:
  - sc_psi, phidsth, and phidcth are computed & returned by the eom_free routine as elements
    [0] and [2] of that routine's work vector argument.
  - Three mode options are available depending on the value of the global flag gf_mag:
    0 - use static dipole (i.e. fixed to earth) with parameters dipole moment, colatitude
        and longitude specified in input file (Lparams.h)
    n - use IGRF2000 internal field model with spherical harmonic terms up to order n (1 to 10)
    yyyy - use static dipole fixed at IGRF2000 yyyy epoch location (valid range 1900-2100)
  - model based on Landau-Lifshitz solution for a spinning sphere in uniform field; recent
    additions attempt to compensate for the over generalization:
    - account for orbit frequency as a linear addition to torque
    - account for cylinder core by making mag. coeff dependent on angle b/w magnetic field and
      cylinder axis (z); see Lparams.h for explanation
    - scale the previous correction b/c cylinder coeff are larger than spherical
    - adjust imaginary mag coeff to account for shell contribution (redundent w/previous)
    -
MODIFICATION HISTORY:
    0210  Scott Williams    Separated this module from deriv() to run as a subroutine
    0211  Scott Williams    Incorporated IGRF2000 Earth magnetic model
    0211  Scott Williams    Added parameterizations to improve on basis LL approach
*******************************************************************************************/
#define R80PI   3.978873577297383394e-03     // =1/80PI   4->4...22209408431
#define R1680PI 1.894701703474944473e-04     // =1/1680PI 3->3...43909242110
#define RMEAN   6.3712e8                     // IGRF specific earth mean radius
void    mag_torques(const double *jd2k, const double *rcube, const double *y,
            const double *sc_psi, const double *phidsth, const double *phidcth, double *Nmag)
{
    int     i;
    double  ai, ar, bL1, bL3, gha, w, zdotB;
    double  work1, work2, work3, work4, work[2];

    gha  = jd2k_2_gha(*jd2k, 0, NULL);

// assumes gf_mag has already been tested for valid mode
// - - - - - - - - - - Use static dipole with user specified moment & location - - - - - - - - -
    if (gf_mag == 0) {
        double  alph, rdotm, sc_alph[2], mdivr3 = g_mdm/(*rcube);

        alph = g_alph0 + gha;           // angular position of z cross m

        // third row of transformation matrix is ECI components of magnetic 'z' axis, i.e., m
        euler_rot(3, g_clatm, alph, 0, g_SCclatm, sc_alph, work, -13, NULL, gv_B);

        // compute magnetic field vector (B) in ECI frame
        gsl_blas_ddot (gv_B, g_orb.v_r, &rdotm);       // rhat dot mhat
        gsl_blas_daxpy (-3*rdotm, g_orb.v_r, gv_B);    // mhat - 3 rdotm rhat
        gsl_vector_scale (gv_B, mdivr3);
    }

//  - - - - - - - - - - - - - - - - - - -  Use IGRF2000 model  - - - - - - - - - - - - - - - - -
    else {
        doublereal    ra, clat, lon, Br, Bt, Bf, r;
        integer       year, nm;

        // - - - - - - - - - -  Use spherical harmonic terms up to order gf_mag  - - - - - - - - -
        if (gf_mag > 0 && gf_mag <= 10)  {
            year = 2000 + floor((*jd2k)/365.25);
            nm   = gf_mag;
        }
        // - - - - - - - - - - - - - - - Use year=gf_mag static dipole - - - - - - - - - - - - - -
        else {
            year = gf_mag;
            nm   = 1;
        }

        // satellite right ascension & longitude
        ra   = atan2(gsl_vector_get (g_orb.v_r, 1), gsl_vector_get (g_orb.v_r, 0));
```

```
        lon  = ra - gha;

        // colatitude of satellite (= pi/2 - declination)
        clat = M_PI_2 - asin(gsl_vector_get (g_orb.v_r, 2));

        r    = g_orb.r/RMEAN;                      // convert radial distance to earth radii

        /* get field components and convert to cartesian coordinates;  IGRF takes discrete
           years as an input to reduce frequency of coefficient determination; cost of
           fractional year interpolation not rewarded with significant increase in accuracy */
        igrf(&year, &nm, &r, &clat, &lon, &Br, &Bt, &Bf);
        bspcar(&clat, &ra, &Br, &Bt, &Bf, &gv_B->data[0], &gv_B->data[1], &gv_B->data[2]);

        // convert from nanoTesla (nT) to guass
        gsl_vector_scale (gv_B, 1.0e-5);
    }

//  - - - - - - - - - - - - -  Now use B to complete the derivations  - - - - - - - - - - - - -
    // transform magnetic field vector to body frame (Goldstein 4-46)
    gsl_blas_dgemv (CblasNoTrans, 1.0, gT_e2b, gv_B, 0.0, gv_Bb);
    // oblate spheroid correction (transfrom from body 'oblate' to body 'sphere')
    gsl_vector_set (gv_Bb, 2, g_oblcore*gsl_vector_get(gv_Bb, 2));
    // compute direction cosine b/w body z axis and B
    if (g_magscl != -999.0) zdotB = gsl_vector_get(gv_Bb, 2) / gsl_blas_dnrm2(gv_Bb);

    for (i=0; i<3; i++) Nmag[i]=0;
    for (i=0; i<=gf_mfreq; i++) {

        if (i == 0) {
            /* compute body frame angular velocity vector & magnitude; store corresponding unit
               vector in 3rd row of gT_b2L matrix--this is the LL z axis                          */
            gsl_vector_set (&gT_b2Lr3.vector, 0, (*phidsth)*sc_psi[0] + y[4]*sc_psi[1]);
            gsl_vector_set (&gT_b2Lr3.vector, 1, (*phidsth)*sc_psi[1] - y[4]*sc_psi[0]);
            // third element includes oblate spheroid correction
            gsl_vector_set (&gT_b2Lr3.vector, 2, g_oblcore*((*phidcth) + y[6]));
            w = gsl_blas_dnrm2(&gT_b2Lr3.vector);
            gsl_vector_scale(&gT_b2Lr3.vector, 1.0/w);
        } else {
            w = g_orbw;
            gsl_vector_set (&gT_b2Lr3.vector, 0, g_orb.sci[0]*g_orb.scW[0]);
            gsl_vector_set (&gT_b2Lr3.vector, 1, g_orb.sci[0]*g_orb.scW[1]);
            // incorporate oblate spheroid correction
            if (g_oblcore!=1.0) {
                // third element includes oblate spheroid correction
                gsl_vector_set (&gT_b2Lr3.vector, 2, g_oblcore*g_orb.sci[1]);
                w *= gsl_blas_dnrm2(&gT_b2Lr3.vector);
                gsl_vector_scale(&gT_b2Lr3.vector, 1.0/w);
            } else  gsl_vector_set (&gT_b2Lr3.vector, 2, g_orb.sci[1]);
        }

        // compute LL y axis--omega cross B (body frame)--and store in 2nd row of gT_b2L
        vector_cross (&gT_b2Lr3.vector, gv_Bb, &gT_b2Lr2.vector);
        work1 = gsl_blas_dnrm2(&gT_b2Lr2.vector);
        gsl_vector_scale(&gT_b2Lr2.vector, 1.0/work1);

        // compute LL x axis from previous results to complete the right handed set
        vector_cross(&gT_b2Lr2.vector, &gT_b2Lr3.vector, &gT_b2Lr1.vector);

        /* transform magnetic field vector to LL frame; note by construction, LL frame y
           y component of B is zero so only need to compute x & z components */
        gsl_blas_ddot (&gT_b2Lr1.vector, gv_Bb, &bL1);
        gsl_blas_ddot (&gT_b2Lr3.vector, gv_Bb, &bL3);

        // compute coefficients of magnetization (eqns 44-47)
        work4 = sqrt(w);
        work1 = g_magcol*work4;                   // 2A*sqrt(2pi*sigma*w)/c
        if (work1 < 0.035) {                      // use low freq approx
            work1 = gsl_pow_2(work1);
            ai   = R80PI*work1;
            ar   = R1680PI*gsl_pow_2(work1);
        } else {
            work2 = sinh(work1);
            work3 = sqrt(1.0 + work2*work2);      // cosh(x) is always positive
            sincos(work1, work);
```

```
            work3 = work3 - work[1];              // denominator term
            work4 = g_magco2/work4;
            ar    = -M_3_8PI     + work4*(work2 - work[0]) / work3;
            ai    = (g_magco3/w) + work4*(work2 + work[0]) / work3;
        }


       /* for a cylinder, ar & ai are twice as big when B is perpindicular to the cylinder axis
          than when parallel, apply this idea to the spherical coefficients as a 'correction' */
        if (g_magscl != -999.) {
            zdotB = g_magfac * (1.0 - 0.5 * fabs(zdotB));
            //zdotB = g_magfac * (0.5 + 0.5 * fabs(zdotB));
            ar  *= zdotB;
            ai  *= zdotB;
        }

        // magnetization coefficient adjustment to account for contribution from spherical shell
        if (g_magshl > 1.0)   ai *= g_magshl;

        // compute the components of torque in the LL frame (eqn 41)
        work1  = g_vcore*bL1;
        gsl_vector_set(gv_work, 2, -work1*ai*bL1);
        work1 *= bL3;
        gsl_vector_set(gv_work, 0,  work1*ai);
        gsl_vector_set(gv_work, 1, -work1*ar);

        /* transform components of torque to body frame (3rd element includes oblate spheroid
           correction, i.e., transform from body spherical to body oblate)                    */
        gsl_blas_dgemv (CblasTrans, 1.0, gT_b2L, gv_work, 0.0, gv_Nmagb);
        Nmag[0] += gsl_vector_get(gv_Nmagb, 0);
        Nmag[1] += gsl_vector_get(gv_Nmagb, 1);
        Nmag[2] += gsl_vector_get(gv_Nmagb, 2)/g_oblcore;
    }
}
#undef R80PI
#undef R1680PI
#undef RMEAN

/*****************************************************************************************
PROGRAM: sat_posn
  Computes Lageos' orbital position given JD2K, a J2000 referenced Julian Date including
  fractional day (i.e. JD - J2000).
INPUTS/OUTPUTS/RETURN VALUE: see above
INCLUDES/EXTERNAL REFERENCES:
  Lincl.h   : math.h, gsl_poly.h, M_2PI
  - Lprot.h : euler_rot()
  Lextern.h : g_orb, g_orbp,
COMMENTS:
  - Equations based on extensive data analysis of NORAD 2 Line Element Sets (2LES) for Lageos 1
  - The orbit plane angular position can be more accurately computed as the net angular
    position with respect to the ascending node (i.e., M+w) as is implimented here.  However,
    mean anomaly is required as a proxy for eccentric anomaly which, in turn, is required
    to determine the instantaneous orbital radius.
  - M+w is much better approximated by a quadratic as opposed to a straight line; however, a
    periodic (non-secular) error remains.  Thus, an option exists to include an additional
    sinusoidal correction term to the M+w calculation for high accuracy needs.
*****************************************************************************************/
void    sat_postn(const double jd2k, struct orb_posn * orb)
{
    // store epoch value
    orb->jd2k = jd2k;

    // determine net angular position wrt ascending node & use sin correction if requested
    orb->Mw   = gsl_poly_eval (g_orbp.Mw, 3, jd2k);
    if (g_orbp.f_sin)
        orb->Mw += g_orbp.Mw[3]*sin((jd2k + g_orbp.Mw[5])*g_orbp.Mw[4]);
    orb->rev = (long) (orb->Mw)/M_2PI;
    orb->Mw  = fmod(orb->Mw, M_2PI);

    // determine RAAN, mean anomaly, eccentric anomaly, & argument of perigee
    orb->W    = gsl_poly_eval (g_orbp.W,  2, jd2k); // right ascension of the ascending node
    orb->M    = gsl_poly_eval (g_orbp.M,  3, jd2k); // mean anomaly
    orb->M    = fmod(orb->M, M_2PI);
    if (orb->M < 0) orb->M += M_2PI;
```

```
    orb->w    = orb->Mw - orb->M;
    if (orb->w < 0) orb->w += M_2PI;
    orb->E    = orb->M + g_orbp.e * sin(orb->M);

    // compute radius
    orb->r    = g_orbp.a * (1.0 - g_orbp.e * cos(orb->E));

    euler_rot(15, g_orbp.i, orb->W, orb->Mw, orb->sci, orb->scW, orb->scMw, -1, NULL, orb->v_r);
}
//*********************************************************************************************
```

LTOOLS.C

```
#include    "Lincl.h"

/*********************************************************************************************
PROGRAM:  elapsed_time
  Computes program execuation duration in minutes and seconds (including fractional seconds)
INPUTS/OUTPUTS/RETURN VALUE: see above
INCLUDES/EXTERNAL REFERENCES:
  Lincl.h : time.h
*********************************************************************************************/
void    elapsed_time(long *eminutes, double *eseconds)
{
    long   cls;
    static unsigned long  clpm = 60*CLOCKS_PER_SEC;

    *eminutes = clock();
    cls       = *eminutes % clpm;
    *eminutes = (*eminutes - cls) / clpm;
    *eseconds = ((double) cls)/((double) CLOCKS_PER_SEC);
}
/*********************************************************************************************
PROGRAM: euler_rot
  Computes transformation matrix between reference frames related by Euler angle rotations,
  designated respectively as 'fixed' and 'rotated.'  Routine actually has two products:
  1) computation and return (if necessary) of sin & cos of the euler angles
  2) the transformation matrix FROM the fixed TO the rotated frames: v_rot = T_tr*v_fix
     *** orthogonal transformation so the inverse is the transpose: v_fix = T_tr'*v_rot
  See comments below for options in handling these products
INCLUDES/NON ANSI: math.h, gsl_matrix.h, FLAG, sincos()
COMMENTS:
  - f_mode specifies whether it is necessary to compute sin & cos of each angle as follows
    if 2 divides f_mode then compute sin & cos of th
    if 3 divides f_mode then compute sin & cos of phi
    if 5 divides f_mode then compute sin & cos of psi
    NOTES: - all possible combinations are achieved with f_modes 0, 1, 2, 3, 5, 6, 10, & 15
           - it's not necessary to provide the angles themselves if sin & cos are already known
  - f_trans specifies output options for the transformation matrix:
    0  = do not compute
    4  = compute entire matrix T_tr s.t. v_rot = T_tr*v_fix
    +i = compute only ith column of T_tr (corresponds to T_tr*ei, ei is the ith basis vector)
    -i = compute only ith row of T_tr (corresponds to ei'*T_tr)
    1* = prepending the value with 1 (i.e., 14, +1i, -1i) will perform the same computations
         under the assumption psi = 0
*********************************************************************************************/
void    euler_rot(const FLAG f_mode, const double th, const double phi, const double psi,
            double *sc_th, double *sc_phi, double *sc_psi, const FLAG f_tr,
            gsl_matrix * T_tr, gsl_vector * v_tr)
{
    double  work1;

    // compute sine & cosine of Euler Angles if necessary
    if (f_mode == 0) {                          // most used case - isolate to speed up
        sincos(th, sc_th);
        sincos(phi, sc_phi);
        sincos(psi, sc_psi);
    } else if (f_mode != 1) {
        if (!(f_mode % 2))  sincos(th, sc_th);
        if (!(f_mode % 3))  sincos(phi, sc_phi);
```

```
        if (!(f_mode % 5))  sincos(psi, sc_psi);
    }

    // compute transformation matrix from fixed to rotated frame if necessary
    switch (f_tr)
    {
      case 0: break;
      case 4:    {   // compute entire matrix
          work1 = sc_th[1]*sc_phi[0];
          gsl_matrix_set (T_tr, 0, 0,  sc_psi[1]*sc_phi[1] - work1*sc_psi[0]);
          gsl_matrix_set (T_tr, 1, 0, -sc_psi[0]*sc_phi[1] - work1*sc_psi[1]);
          gsl_matrix_set (T_tr, 2, 0,  sc_th[0]*sc_phi[0]);
          work1 = sc_th[1]*sc_phi[1];
          gsl_matrix_set (T_tr, 0, 1,  sc_psi[1]*sc_phi[0] + work1*sc_psi[0]);
          gsl_matrix_set (T_tr, 1, 1, -sc_psi[0]*sc_phi[0] + work1*sc_psi[1]);
          gsl_matrix_set (T_tr, 2, 1, -sc_th[0]*sc_phi[1]);
          gsl_matrix_set (T_tr, 0, 2,  sc_th[0]*sc_psi[0]);
          gsl_matrix_set (T_tr, 1, 2,  sc_th[0]*sc_psi[1]);
          gsl_matrix_set (T_tr, 2, 2,  sc_th[1]);
          break; }
      case 1:    {   // compute 1st column only: T_tr*e1
          work1 = sc_th[1]*sc_phi[0];
          gsl_vector_set (v_tr, 0,  sc_psi[1]*sc_phi[1] - work1*sc_psi[0]);
          gsl_vector_set (v_tr, 1, -sc_psi[0]*sc_phi[1] - work1*sc_psi[1]);
          gsl_vector_set (v_tr, 2,  sc_th[0]*sc_phi[0]);
          break; }
      case 2:    {   // compute 2nd column only: T_tr*e2
          work1 = sc_th[1]*sc_phi[1];
          gsl_vector_set (v_tr, 0,  sc_psi[1]*sc_phi[0] + work1*sc_psi[0]);
          gsl_vector_set (v_tr, 1, -sc_psi[0]*sc_phi[0] + work1*sc_psi[1]);
          gsl_vector_set (v_tr, 2, -sc_th[0]*sc_phi[1]);
          break; }
      case 3:    {   // compute 3rd column only: T_tr*e3
          gsl_vector_set (v_tr, 0, sc_th[0]*sc_psi[0]);
          gsl_vector_set (v_tr, 1, sc_th[0]*sc_psi[1]);
          gsl_vector_set (v_tr, 2, sc_th[1]);
          break; }
      case -1:   {   // compute 1st row only: e1'*T_tr
          work1 = sc_th[1]*sc_psi[0];
          gsl_vector_set (v_tr, 0, sc_psi[1]*sc_phi[1] - work1*sc_phi[0]);
          gsl_vector_set (v_tr, 1, sc_psi[1]*sc_phi[0] + work1*sc_phi[1]);
          gsl_vector_set (v_tr, 2, sc_th[0]*sc_psi[0]);
          break; }
      case -2:   {   // compute 2nd row only: e2'*T_tr
          work1 = sc_th[1]*sc_psi[1];
          gsl_vector_set (v_tr, 0, -sc_psi[0]*sc_phi[1] - work1*sc_phi[0]);
          gsl_vector_set (v_tr, 1, -sc_psi[0]*sc_phi[0] + work1*sc_phi[1]);
          gsl_vector_set (v_tr, 2,  sc_th[0]*sc_psi[1]);
          break; }
      case -3:   {   // compute 3rd row only: e3'*T_tr
          gsl_vector_set (v_tr, 0,  sc_th[0]*sc_phi[0]);
          gsl_vector_set (v_tr, 1, -sc_th[0]*sc_phi[1]);
          gsl_vector_set (v_tr, 2,  sc_th[1]);
          break; }
// **** Same as above but with assumption psi=0 ****
      case 14:    {   // compute entire matrix
          gsl_matrix_set (T_tr, 0, 0,  sc_phi[1]);
          gsl_matrix_set (T_tr, 1, 0, -sc_th[1]*sc_phi[0]);
          gsl_matrix_set (T_tr, 2, 0,  sc_th[0]*sc_phi[0]);
          gsl_matrix_set (T_tr, 0, 1,  sc_phi[0]);
          gsl_matrix_set (T_tr, 1, 1,  sc_th[1]*sc_phi[1]);
          gsl_matrix_set (T_tr, 2, 1, -sc_th[0]*sc_phi[1]);
          gsl_matrix_set (T_tr, 0, 2,  0);
          gsl_matrix_set (T_tr, 1, 2,  sc_th[0]);
          gsl_matrix_set (T_tr, 2, 2,  sc_th[1]);
          break; }
      case 11:    {   // compute 1st column only: T_tr*e1
          gsl_vector_set (v_tr, 0,  sc_phi[1]);
          gsl_vector_set (v_tr, 1, -sc_th[1]*sc_phi[0]);
          gsl_vector_set (v_tr, 2,  sc_th[0]*sc_phi[0]);
          break; }
      case 12:    {   // compute 2nd column only: T_tr*e2
          gsl_vector_set (v_tr, 0,  sc_phi[0]);
          gsl_vector_set (v_tr, 1,  sc_th[1]*sc_phi[1]);
```

```
          gsl_vector_set (v_tr, 2, -sc_th[0]*sc_phi[1]);
          break; }
      case 13:    {   // compute 3rd column only: T_tr*e3
          gsl_vector_set (v_tr, 0, 0);
          gsl_vector_set (v_tr, 1, sc_th[0]);
          gsl_vector_set (v_tr, 2, sc_th[1]);
          break; }
      case -11:   {   // compute 1st row only: e1'*T_tr
          gsl_vector_set (v_tr, 0, sc_phi[1]);
          gsl_vector_set (v_tr, 1, sc_phi[0]);
          gsl_vector_set (v_tr, 2, 0);
          break; }
      case -12:   {   // compute 2nd row only: e2'*T_tr
          gsl_vector_set (v_tr, 0, -sc_th[1]*sc_phi[0]);
          gsl_vector_set (v_tr, 1,  sc_th[1]*sc_phi[1]);
          gsl_vector_set (v_tr, 2,  sc_th[0]);
          break; }
      case -13:   {   // compute 3rd row only: e3'*T_tr
          gsl_vector_set (v_tr, 0,  sc_th[0]*sc_phi[0]);
          gsl_vector_set (v_tr, 1, -sc_th[0]*sc_phi[1]);
          gsl_vector_set (v_tr, 2,  sc_th[1]);
          break; }
    }
}

/*************************************************************************************************
PROGRAM:  expcat
  Catenates a double in exponential format to remove extra zero digits from the exponent.  The
  base is the return value and the exponent is stored in the integer exp.
INPUTS/OUTPUTS/RETURN VALUE: see above
INCLUDES/EXTERNAL REFERENCES:
  Lincl.h : stdio.h, string.h, stdlib.h
COMMENTS:
*************************************************************************************************/
double  expcat(double x, int *expn)
{
    char  str[30], *strptr;

    sprintf(str,"%.16e", x);
    strptr  = strpbrk (str, "e");
    *strptr = '\0';
    strptr += 1;
    *expn = atoi(strptr);
    return strtod (str, &strptr);
}

/*************************************************************************************************
PROGRAM:  jd2k_2_gha
  Computes time of day Greenwich Hour Angle given JD2K - a J2000 referenced Julian Date
  including fractional day (i.e. JD - J2000).  Optionally returns GHA at 0h as well (f_gh0!=0).
INPUTS/OUTPUTS/RETURN VALUE: see above
INCLUDES/EXTERNAL REFERENCES:
  Lincl.h : math.h, FLAG, RPJS
COMMENTS:
   Direct source: T.S. Kelso, "Orbital Coordinate Systems, Part II," Satellite Times,
  www.celestrak.com; Original source: Explanetory Supplement to the Astronomical Almanac.
*************************************************************************************************/
#define RPJS   7.272205216643039904e-05     // =2PI/86400  4->3...84871153537
#define FACT   8.663655366976000000e04      // =86400*1.00273790934
double  jd2k_2_gha(const double jd2k, const FLAG f_gh0, double *gh0)
{
    double ut, tu, gha;

    ut   = jd2k+0.5;
    ut   = ut - floor(ut);                          // fractional part of day from 0h
    tu   = jd2k - ut;                               // JD2K at 0h of jd2k
/*  tu   = tu/36525;                                // Julian Centuries from J2000 to 0h jd2k
    gha = 24110.54841 + tu*(8640184.812866 + tu*(0.093104 - 6.2e-6*tu));
    divided poly coefficients by powers of 36525 to eliminate unneccessary flop
*/
    gha = 24110.54841 +
          tu*(236.55536790872 + tu*(6.978914707327780e-11 - 1.272392513927221e-19*tu));
    if (f_gh0) *gh0 = RPJS*fmod(gha,86400);
    gha = gha + FACT*ut;
```

```
    gha = RPJS*fmod(gha,86400);

    return gha;
}
#undef RPJS
#undef FACT
/*************************************************************************************************
PROGRAM:  sincos
  Computes sine (sc_x[0]) and cosine (sc_x[1]) of an angle (x) using only one trig evaluation
INPUTS/OUTPUTS/RETURN VALUE:  see above
INCLUDES/EXTERNAL REFERENCES:
  Lincl.h : math.h, gsl_math.h, M_2PI
*************************************************************************************************/
void    sincos(double x, double *sc_x)
{
    if (x > M_2PI || x < -M_2PI)  x = fmod(x, M_2PI);
    sc_x[1] = cos(x);
    sc_x[0] = sqrt(1.0 - gsl_pow_2(sc_x[1]));
    if (x > M_PI || (x < 0 && x > -M_PI)) sc_x[0] = -sc_x[0];
}

/*************************************************************************************************
PROGRAM:  vector_cross
  Computes the cross product of gsl type vectors a & b and returns gsl type vector c = a x b
INPUTS/OUTPUTS/RETURN VALUE: see above
INCLUDES/EXTERNAL REFERENCES:
  Lincl.h : math.h, gsl_vector.h
*************************************************************************************************/
void    vector_cross(const gsl_vector * a, const gsl_vector * b, gsl_vector * result)
{
    gsl_vector_set(result, 0,
        gsl_vector_get(a,1)*gsl_vector_get(b,2) - gsl_vector_get(a,2)*gsl_vector_get(b,1));
    gsl_vector_set(result, 1,
        gsl_vector_get(a,2)*gsl_vector_get(b,0) - gsl_vector_get(a,0)*gsl_vector_get(b,2));
    gsl_vector_set(result, 2,
        gsl_vector_get(a,0)*gsl_vector_get(b,1) - gsl_vector_get(a,1)*gsl_vector_get(b,0));
}
//*************************************************************************************************
```

LPARAMS.H

```
/*~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ LPARAMS.H ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
  Lparams.h is the 'control center' for the Lageos spin model.  All input data of consequence
  is defined here (with references to the routines that make use of the values).  This includes
  integrator control parameters, physical and mathematical constants describing satellite
  properties and the space environment, and, of course, the initial system state.
-------------------------------------------------------------------------------------------------*/
#define TITLE  "Lageos Spin Dynamics Model Version 5.0\0"
#define DESCR  "Data Analysis\0"
#define INSRC  "Program Defaults\0"

/*~~~~~~~~~~~~~~~~~~~~~~~~~~ integrator control & variable conditioning ~~~~~~~~~~~~~~~~~~~~~~~~~~~
  General control parameters for lageos_spin_xx() driver routines in lmain.c.  Usage of these
  parameters varies with the integration routine; some values may not be (explicitly) used
  by a given routine.
-------------------------------------------------------------------------------------------------*/
#define _TINY    1.0e-30          // safety factor - prevents divide by zero
#define _HMIN    5.0e-16          // minimum relative step size allowed
#define _NVAR    6                // number of dependent variables
#define RTOL     1.0e-12          // relative error tolerance
#define ATOL     GSL_DBL_EPSILON  // absolute error tolerance

#define START    -2826.333206     // Start time reference date (JD2K, i.e., JD - J2000)
                                  // -2712.420775 = 29 Jul 02 @ 01:54:05Z
#define STOP     -2240.446528     // Stop time reference date (JD2K)
#define H1       HMAX             // initial integration step size (s)
#define HMAX     100.0            // max step size (s) allowed; 13.5~1/1000 orbit: rel dB < 2%
#define MAXSTP   LONG_MAX         // max number of integration steps

#define RESET    1.0e7            // max value (s) of local indep variable b4 reset to zero
```

```
#define PHIMOD   4               // modulo euler angle phi to [0, PHIMOD*2pi]
#define PSIMOD   4               // modulo euler angle psi to [0, PSIMOD*2pi]


/*~~~~~~~~~~~~~~~~~~~~~~~~~~~~ data set generation & output control ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
  2 modes are available for data set generation:
   - "Recurrent" writes data sets to output files on the periodic interval SAVE; SAVE=0 disables
     this option, though the initial and final data sets are always written to the output files.
   - "targeted" generates data sets per FOUT at the specific JD2K times listed in OUT1...OUT5;
     the sets are stored in internal arrays by default (e.g. to interact w/external routines)
     FOUT: 0=disabled, 1=write to files, 2=internal store only (no file write )
   IMPORTANT: to compile, OUT1...OUT6 must be non-empty and _NOUT must always equal their total
     number of values, even if FOUT=0.
   IMPORTANT2: In the output files, all angular measures are in degrees; however, the internal
     storage for targeted outputs keeps angles in radians
-----------------------------------------------------------------------------------------------*/
#define SAVE     0.0             // "recurring" data output interval (JD2K), 0 for none
#define FOUT     0               // "targeted" data output times flag (see above)
#define _NOUT    32              // number of target output times listed below (size of array)
#define OUT1     -4110.109722,-4040.265278,-3783.213194,-2826.333206,-2772.093322
#define OUT2     -2769.118056,-2761.291470,-2758.314155,-2712.420775,-2678.446690
#define OUT3     -2650.489155,-2647.443750,-2642.493750,-2633.503472,-2625.526389
#define OUT4     -2443.370139,-2439.306250,-2430.370139,-2404.369444,-2359.197917
#define OUT5     -2299.422222,-2268.318750,-2241.531944,-2240.446528,-1771.164583
#define OUT6     -1755.327083,-1732.318056,-1284.500000,-1172.500000,0.0,0.643799,25000


/*~~~~~~~~~~~~~~~~~~~~~~~~~~~~ model implementation option switches ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
  Flags to specify usage of specific model components/options
-----------------------------------------------------------------------------------------------*/
#define _FOPT    0      /* parameter optimization : 0=std data run; 1=param opt; 2=std+opt    */
#define DRIVER   3      /* lageos_spin_xx() : 0=_nr bs; 1=_nr bd; 2=_rk; 3=_de
                           Note: _de not recommended w/small values of RESET; see
                                 lageos_spin_de() header comments                            */
#define FETASIN  0      /* Orbit model net angular position option flag: Use high accuracy
                           sinusoidal correction for M+w? 0=no, else yes                       */
#define FGRAV    1      /* Gravity model option flag: Include 1st zonal term (J2) in torque
                           calculation? 0=no, else yes                                         */
#define FMAG     3       /* Magnetic model option mode flag:
                           0      = use static dipole given by MDM, CLATM, & LONM below
                           yyyy   = use year yyyy static dipole from IGRF2000 (1965 to 2005)
                           1...10 = use IGRF model with spherical harmonic terms up to order
                                    FMAG; i.e., 1=dipole, 2=quadrupole, 3=octupole,...;       */
#define FMFREQ   1      /* Specifies whether to include orbit frequency in computation of
                           magnetic torque (as an additive correction): 0-no; else yes        */


/*~~~~~~~~~~~~~~~~~~~ physical/geophysical (Earth) constants and parameters ~~~~~~~~~~~~~~~~~~~~~
  C1, GM, J2, RE, and RFLAT taken from IERS Conventions 2000 (draft).  The mean earth radius
  (derived from RE & RFLAT) and the IGRF 2000 magnetic model are used to generate a dipole
  approximation (MDM, CLATM, LONM) for 1995; see FMAG comments above.
-----------------------------------------------------------------------------------------------*/
#define C1       2.99792458e10    // speed of light (cm/s)
#define GM       3.986004418e20   // mass of the earth times G (cm^3/s^2)
#define J2       1.0826359e-3     // 1st order oblate earth zonal coefficient
#define RE       6.3781366e8      // equatorial radius of the earth (cm)
//#define RFLAT    298.25642        // reciprocal earth flattening coefficient (i.e. = 1/f)
#define MDM      7.8115998e25     // earth magnetic dipole moment (gauss*cm^3)
#define CLATM    10.7044330       // co-latitude (pi/2 - lat) of magnetic dipole (deg)
#define LONM     -71.4068205      // longitude of magnetic dipole (deg)


/*~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ satellite orbit parameters ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
  These parameters describe Lageos' orbital motion.  The elements were derived from the NORAD
  2 Line Element Sets (2LES) for Lageos 1 dating back to 1980.  The regularity of Lageos' orbit
  makes for very stable orbit elements, however, they are not technically constant.  Various
  regression techniques were used to obtain 'best fit' functions to the data.  The results
  lead readily to choices for constants as approximations.  The current data set was derived as
  follows:
  - Orbit semi-major axis (A) - derived from mean motion using Kepler's equations
  - Eccentricity (ECC) - mean value of all historic data
  - Inclination (INC) - mean value of >=1990 periodic data
  - Orbital Precession (RAAND, RAAN0) - linear model
  - angular position (ETA..) : quadratic model with optional sinusoidal correction (for
```

```
    high accuracy); Note: eta = mean anomaly + argument of perigee
  - Mean Anomaly (M..) : quadratic model; Note - only used to compute instantaneous radius;
    M+w gives a more accurate angular position
-------------------------------------------------------------------------------------------*/
#define A        1.227119174e9          // semi-major axis of the orbit (cm)
#define ECC      0.0044319              // eccentricity of orbit
#define INC      109.84188              // inclination of orbit (deg)
#define RAAN0    109.051226             // J2000 right ascension of ascending node (deg)
#define RAAND    0.3425558365501        // orbital precession rate (deg/JD)
#define ETA0     319.420422218739       // Quad: J2000 satellite angular position (deg)
#define ETAD     2298.97906232758       // Quad: net angular motion of satellite (deg/JD)
#define ETADD    1.77236870513298e-7    // Quad: angular acceleration (deg/JD^2)
#define ETAM     2.56071177510377e-2    // Sin : magnitude (deg)
#define ETAA     0.23527273308971       // Sin : amplitude (deg)
#define ETAT     1061.46420515695       // Sin : period (JD)
#define ETAP     291.566705595897       // Sin : phase shift (JD)
#define M0       107.570967400446       // Quad: J2000 Mean Anomaly (deg)
#define MD       2299.19307317095       // Quad: Mean motion term (deg/JD)
#define MDD      1.70332256743677e-7    // Quad: acceleration term (deg/JD^2)


/*~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ satellite parameters ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
  These parameters define the satellite properties for the model.
  RCORE, SIGC  - define the reference homogeneous metallic spheroid for the purpose of
                 modelling magnetic torques.  The reference spheroid is intended to approximate
                 only the effects induced on the cylindrical core of Lageos.

  FLATC        - added as of v4.1 to allow the core to be modeled as an oblate spheroid
                 rather than purely spherical. It should be noted that the implimentation
                 is somewhat crude and is intended only to parameterize the gap between the
                 purely spherical Landau-Lifshitz development and the true cylindrical core;
                 better is the implementation in MCSCL below.  The transformation is simply a
                 scaling of the z components of B, w, and (inversely) the resulting N by the
                 ratio of the polar to equatorial radius of the spheroid.

   MCSCL       - the magnetization coefficients for a cylinder, ar & ai, are twice as big
                 when B is perpindicular to the cylinder axis (z) as when parallel; this idea
                 is adapted to the spherical magnetization coefficients of the current model
                 as a 'correction'.  The relative scaling is done as a function of the
                 direction cosine between the body z-axis and the magnetic field.  The ratios
                 are fixed but there is still freedom to choose the absolute scaling (MCSCL)
                 of the coefficients.

                 Implementation:
                      ar' = MCSCL*(1-0.5 |z dot B|)ar    ==>   0.5*MCSCL <= ar' <= MCSCL
                      ai' = MCSCL*(1-0.5 |z dot B|)ai    ==>   0.5*MCSCL <= ai' <= MCSCL
                where ar and ai are the "pure" spherical versions of the magnetization
                coefficients.  The realtive range of the scaled coefficients is shown with
                the minimums occuring when B parallel to z (cylinder axis) and maximums
                when B perpindicular to z.  To turn this feature off, set MCSCL = -999.  To
                ensure the "pure" values are attained for some |z dot B|, use values b/w
                1 and 2.  Also interesting, the low frequency cylindrical coefficients scale
                to ~2.5 times those for the sphere so this is another interesting value to
                use.

  MCSHL        - A first order approximation of the additional magnetic torque from the shell
                 leads to an expression for net magnetic torque identical to the that of the
                 core except with instances of V*ai replaced with V*ai + k*w where w is the
                 angular frequency and k is a NON-NEGATIVE constant that depends on Vs (volume
                 of shell) and electromagnetic props.

                 It is reasonable to write k*w = mu*Vs*ais where mu a positive proportionality
                 constant and ais is the imag magnetic coefficient of the shell.  With a little
                 algebra, the problem reduces to an additional scaling of the imaginary mag.
                 coeff, i.e., ai -> (1 + MCSHL)*ai and MCSHL is proportional to (Vs/V)*(ais/ai)

                 Further, a separate calculation (based on the Landau-Lifshits solutions) shows
                 the mag coeff of the shell agree with the expression for the mag coeff of the
                 core with one additional term (that is extremely messy so doesn't lead to a
                 usable form).  Thus ais/ai = 1 +/- mu' and mu' will depend on the respective
                 dimensions and effective conductivities.  Given the uncertainties about the
                 current flow in the shell, this is a very 'loose' parameter but it is not a
                 forgone conclusion that mu should be small.
```

```
                NOTE: This factor is basically redundent with MCSCL above and that is
                      favored for the additional dynamics it includes.

  I1 & I3      - are the principle moments of the Lageos satellite.  Common values in the
                 literature are 1.271e8 and 1.314e8 respectively, though modest deviations
                 from these values for optimal performance is expected.

  TUAG         - is a optional scaling factor for the computed gravitational torques; this
                 factor allows 'tweaking' of model for possibly better results.  Values of
                 TAUG = 1 +/- eps (something small) are reasonable.  This may be particularly
                 useful as a low cost substitute for the oblate earth J2 term which scales
                 (more or less) as a small multiple of the primary term.  For paramter
                 optimization, however, it is probably better to vary I1 and I3 and leave
                 this set to 1. (Ed. Note: this tweak largely unused)
-------------------------------------------------------------------------------------------*/
#define RCORE    23.5339          // equatorial radius of sat core reference spheroid (cm)
#define SIGC     1.0e17           // eff. conductivity of sat. core metallic ref sphere (1/s)
#define FLATC    0                // core flattening coefficient = 1 - Rp/Re
#define MCSCL    2.4970           /* scaling factor for magnetization coefficients (see above)
                                  // MCSCL = -999 to disable (other neg values forced to 0)   */
#define MCSHL    0.0              // spherical shell correction factor (see above);
#define I1       1.27978e8        // transverse moment of inertia (g cm^2)
#define I3       1.30701e8        // axial moment of inertia (g cm^2)
#define TAUG     1.0              // gravity torque scaling parameter


/*~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ initial spin state ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
  Spin state at time = START.  Values were derived from Avizonis' empirical 'flash' data; the
  arguments represent the classic Euler Angles relating the body frame to the inertial frame;
  at the current epoch, Lageos' angular velocity is nearly pure axial and so the transvers
  rates (THETAD, PHID) can be considered negligeable without loss of generality
-------------------------------------------------------------------------------------------*/

//-2826.333206
#define THETA    165.70           // deg
#define PHI      3.56             // deg
#define PSI      0.0              // deg
#define THETAD   1e-16            // deg/s
#define PHID     1e-16            // deg/s
#define PSID     3.07640          // deg/s


/*
//-2439.306250
#define THETA    162.90           // deg
#define PHI      178.98            // deg
#define PSI      0.0              // deg
#define THETAD   1e-16            // deg/s
#define PHID     1e-16            // deg/s
#define PSID     2.15827          // deg/s

//-2678.446690
#define THETA    170.67           // deg
#define PHI      90.36            // deg
#define PSI      0.0              // deg
#define THETAD   1e-16            // deg/s
#define PHID     1e-16            // deg/s
#define PSID     2.69865          // deg/s

//-2712.420775
#define THETA    170.60           // deg
#define PHI      44.32            // deg
#define PSI      0.0              // deg
#define THETAD   1e-16            // deg/s
#define PHID     1e-16            // deg/s
#define PSID     2.80003          // deg/s

//-2769.118056
#define THETA    166.86           // deg
#define PHI      34.78            // deg
#define PSI      0.0              // deg
#define THETAD   1e-16            // deg/s
#define PHID     1e-16            // deg/s
#define PSID     2.94046          // deg/s
```

```
-------------------------------------------------------------------------------------------*/
```

Numerical Routines

LDOP853.C & LDOP853.H– external package; see [iii].

LSHODE.F – external package; see [iv].

LNR_C.C – external package (see [ii]) but customized

```
#include    "Lincl.h"

#define    IMAX    11                           // used in bsstep
#define    NUSE    7                            // used in bsstep & rzextr
#define    SHRINK  0.95                         // used in bsstep
#define    GROW    1.2                          // used in bsstep

// Counting number of derivative function calls . . .
static unsigned long nfcn;
unsigned long fcncntRead(void)
{
    return nfcn;
}
void fcncntReset(void)
{
    nfcn = 0;
}
/*******************************************************************************************
PROGRAM:  bsstep
  Uses a modified Burlisch-Stoer extrapolation approach to advance a vector of dependent
  variables a single step (y(x) to y(x+hdid)) while monitoring local truncation error to
  ensure accuracy and adjust step size.  An estimate for the next step to be taken is also
  generated.
INPUTS/OUTPUTS/RETURN VALUE:
- - - - - - - - *** all vectors are assumed unit offset; range [1...nvar] ***  - - - - - - - - -
  y      - vector of dependent variables at xx; replaced with new values on output
  dydx   - vector of derivatives of y at xx
  nv     - length of the vectors
  xx     - value of independent variable; replaced with new value on output
  h      - the step of the indpendent variable to be attempted
  hdid   - the step of the indpendent variable actually accomplished
  hnext  - estimate of the next step size to take
  eps    - relative error tolerance
  yscal  - vector against which error is scaled
  derivs - user supplied function that computes the derivatives (3rd argument) of the dependent
           variables (2nd argument) at a specified value of the independent variable (1st
           argument)
INCLUDES/EXTERNAL REFERENCES:
  Lincl.h : math.h
  - Lprot.h   : lageos_error(), mmid(), rzextr()
  Lintgr8.c : GROW, IMAX, NUSE, SHRINK
COMMENTS:
   Adapted from Numerical Recipes in C, 2nd Ed., pp 728-730; taylored to fit current problem.
  The approach used is from the family of extrapolation methods.  Original NR code used
  adaptive stepsize control scheme suggested by Deuflhard; this adaptation uses a simpler
  method similar to that originally proposed by Bulirsch & Stoerr when they introduced the
  extrapolation approach.
   In the world of ODE methods, Runge-Kutta approaches seem to be more generally favored (ease
  of use, at least as fast, stability) but extrapolation methods have the advantage of being
  variable order and so are suited well for high accuracy needs and long integration times as
```

```
  is the case in the present application.  It should be noted that some efficiency is lost in
  using the simpler stepsize control algorithm but the approach is more intuitive.
MODIFICATION HISTORY:
  ????    Miller & Holz    First Release
  9710    Scott Williams   Cleaned up code; added explanetory commentary
  0210    Scott Williams   Replaced mallocs with VLAs (C99); modified stepsize underflow trap
*******************************************************************************************/
void    bsstep(double *y, const double *dydx, const int nv, double *xx, double h,
            const double eps, const double *yscal, double *hdid, double *hnext,
            void (*derivs)(const double, const double *, double *))
{
    int     i, j=nv+1;
    double  xest, errmax, temp;
    double  ysav[j], yseq[j], yerr[j], x[j], d[j][NUSE+1];
    static int    nseq[IMAX+1]={0,2,4,6,8,12,16,24,32,48,64,96};
    static double nratio[IMAX+1];


    // set stepsize scaling factors on first call to routine
    if (!nratio[1]) {
        for (i=1; i<NUSE-1; i++)      nratio[i] = ((double) nseq[NUSE-1])/((double) nseq[i]);
        nratio[NUSE-1]    = GROW;
        nratio[NUSE]    = SHRINK;
        for (i=NUSE+1; i<=IMAX; i++)  nratio[i] = ((double) nseq[NUSE-1])/((double) nseq[i]);
        nratio[0]        = 0.25;
        for (i=1; i<=((IMAX-NUSE)/2); i++)  nratio[0]  *= 0.5;
    }

    //nfcn = 0;
    for (i=1; i<=nv; i++)    ysav[i] = y[i];       // save input function values
    for (;;) {
        for (i=1; i<=IMAX; i++) {

            // step to xx+h using nseq[i] substeps
            mmid(ysav, dydx, nv, (*xx), h, nseq[i], yseq, derivs);
            xest = (temp=h/nseq[i], temp*temp);    // squared b/c error series even

            // perform extrapolation; result is limit as substep size goes to 0
            rzextr(i, xest, yseq, y, yerr, nv, x, NUSE+1, d);
            errmax= 0.0;
            for (j=1; j<=nv; j++) {                // compute largest relative error
                temp   = fabs(yerr[j]/yscal[j]);
                errmax = (errmax < temp) ? temp : errmax;
            }
            if (errmax/eps < 1.0) {                // acceptible error so good step
                *xx    += h;
                *hdid  = h;
                *hnext = h*nratio[i];              // predict next step
                return;
            }
        }
        h *= nratio[0];                            // step too big, shrink & try again
        if ((*xx+h) == (*xx)) {                    // oops, now too small
            printf("Stepsize underflow in BSSTEP; trying a slightly bigger step . . .\n");
            while ((*xx+h) == (*xx)) h *= 1.1;
        }
    }
}
#undef    IMAX
#undef    SHRINK
#undef    GROW

/*******************************************************************************************
PROGRAM:  mmid
  Uses the 'modified midpoint method' to advance a vector of dependent variables a single step
  (y(x) to y(x+H)) using a sequence of substeps
INPUTS/OUTPUTS/RETURN VALUE:
- - - - - - - - *** all vectors are assumed unit offset; range [1...nvar] ***  - - - - - - - - -
  y      - vector of dependent variables at xs
  dydx   - vector of derivatives of y at xs
  nvar   - length of the vectors
  xs     - initial value of independent variable
  htot   - the total step of the indpendent variable to be made
  nstep  - number of substeps to use
```

```
  yout   - vector of resulting values of y at xs+htot; this may be the same vector as y
  derivs - user supplied function that computes the derivatives (3rd argument) of the dependent
           variables (2nd argument) at a specified value of the independent variable (1st
           argument)
INCLUDES/EXTERNAL REFERENCES:
  Lincl.h : math.h
COMMENTS:
   Adapted from Numerical Recipes in C, 2nd Ed., pp 723-724.  This is a 2nd order 'centered
  difference' method (except at the end points) for integrating ODEs.  It is not very useful
  by itself but it is valuable as part of more powerful methods because its error series only
  even terms.
MODIFICATION HISTORY:
  ????    Miller & Holz    First Release
  9710    Scott Williams   Cleaned up code; added explanetory commentary
  0210    Scott Williams   Replaced mallocs with VLAs (C99)
*******************************************************************************************/
void    mmid(const double *y, const double *dydx, const int nvar, const double xs,
             const double htot, const int nstep, double *yout,
             void (*derivs)(const double, const double *, double *))
{
    int     n=nvar+1, i;
    double  x, swap, h2, h;
    double  ym[n], yn[n];

    h = htot/nstep;                                 // compute substep
    for (i=1; i<=nvar; i++) {
        ym[i] = y[i];
        yn[i] = y[i] + h*dydx[i];                   // first step
    }
    x    = xs+h;
    (*derivs)(x, yn, yout);                         // use yout for temporary storage
    nfcn++;
    h2 = 2.0*h;
    for (n=2; n<=nstep; n++) {                      // general 'centered' steps
        for (i=1;i<=nvar;i++) {
            swap  = ym[i] + h2*yout[i];
            ym[i] = yn[i];
            yn[i] = swap;
        }
        x += h;
        (*derivs)(x, yn, yout);
        nfcn++;
    }
    for (i=1; i<=nvar; i++)                         // compute yout at xs+htot
        yout[i] = 0.5*(ym[i] + yn[i] + h*yout[i]);
}

/*******************************************************************************************
PROGRAM:  rzextr
  Uses rational function extrapolation (ratio of polynomials) to project a 'function' value at
  x = 0, y(0), using a sequence of estimates generated from progessively smaller values of x
  and corresponding values y(x).  y may be a vector functions (i.e., y = [y1(x), . . .,yn(x)].
INPUTS/OUTPUTS/RETURN VALUE:
- - - - - - - - *** all vectors are assumed unit offset; range [1...nvar] ***  - - - - - - - - -
  iest - index of current (newest) set of values in sequence values
  xest - current (newest=smallest) value of independent variable
  yest - current (newest) set of function values at xest
  yz   - resulting vector of current extrapolated function values at x=0
  dy   - vector of estimated errors associated with for yz
  nv   - number of components in the vector function y(x)
  x    - vector containing previous sequence of xest {x1, . . ., x(iest-1)}
  d    - matrix containing previous sequence of yest {y(x1), . . ., y(x(iest-1))}
         NOTE: x & d are appended with xest & yest respectively on output
INCLUDES/EXTERNAL REFERENCES:
  Lincl.h : math.h
  Lintgr8.c : NUSE
COMMENTS:
   Adapted from Numerical Recipes in C, 2nd Ed., pp 731-732.  Implemented version imposes
    a limit (NUSE) on the number of values to be used in the extrapolation.
MODIFICATION HISTORY:
  ????    Miller & Holz    First Release
  9710    Scott Williams   Cleaned up; added commentary, set x & d as arguments vice globals
  0210    Scott Williams   Replaced mallocs with VLAs (C99)
*******************************************************************************************/
```

```
void    rzextr(const int iest, double xest, const double *yest, double *yz, double *dy,
           const int nv, double *x, const int nc, double d[nv+1][nc])
{
    int        m1, k, j;
    double    yy, v, ddy, c, b1, b;

    x[iest] = xest;                                 // save current indep variable
    if (iest == 1)    {                             // 1st time so can't extrapolate
        for (j=1; j<=nv; j++) {
            yz[j]   = yest[j];
            d[j][1] = yest[j];
            dy[j]   = yest[j];
        }
    } else {
        double fx[NUSE+1];
        m1   = (iest<NUSE) ? iest : NUSE;           // use last NUSE sequence values
        xest = 1.0/xest;                            // multiply faster than divide
        for (k=1; k<m1; k++)
            fx[k+1] = x[iest-k]*xest;               // normalize to current xest
        for (j=1; j<=nv; j++) {                     // get next diagonal in tableau
            yy      = yest[j];
            v       = d[j][1];
            c       = yy;
            d[j][1] = yy;
            for (k=2; k<=m1; k++) {
                b1 = fx[k]*v;
                b  = b1-c;
                if (b) {
                    b   = (c-v)/b;
                    ddy = c*b;
                    c   = b1*b;
                } else                              // precaution against divide by 0
                    ddy = v;
                if (k != m1) v = d[j][k];
                d[j][k] = ddy;
                yy += ddy;
            }
            dy[j] = ddy;
            yz[j] = yy;
        }
    }
}

#undef    NUSE

/*************************************************************************************************
PROGRAM:  bdstep
  The orginal Numerical Recipes version of bsstep . . . Uses the extrapolation method of
  Bader & Deuflhard to advance a vector of dependent variables a single step (y(x) to
  y(x+hdid)) while monitoring local truncation error to ensure accuracy and adjust step size.
  An estimate for the next step to be taken is also generated.
INPUTS/OUTPUTS/RETURN VALUE:
- - - - - - - - *** all vectors are assumed unit offset; range [1...nvar] ***  - - - - - - - - -
  y      - vector of dependent variables at xx; replaced with new values on output
  dydx   - vector of derivatives of y at xx
  nv     - length of the vectors
  xx     - value of independent variable; replaced with new value on output
  htry   - the step of the indpendent variable to be attempted
  hdid   - the step of the indpendent variable actually accomplished
  hnext  - estimate of the next step size to take
  eps    - relative error tolerance
  yscal  - vector against which error is scaled
  derivs - user supplied function that computes the derivatives (3rd argument) of the dependent
           variables (2nd argument) at a specified value of the independent variable (1st
           argument)
INCLUDES/EXTERNAL REFERENCES:
  Lincl.h : math.h, gsl_math.h, _TINY
  - Lprot.h   : lageos_error(), mmid(), rzextr()
  Lintgr8.c : IMAXX, KMAXX, REDMAX, SAFE, SAFE2, SCALMX,
COMMENTS:
   Adapted almost verbatim from Numerical Recipes in C, 2nd Ed., pp 728-730.  The approach
  used is from the family of extrapolation methods introduced by Bulisrch & Stoerr.  The
  adaptive stepsize control scheme is that of Deuflhard and it claims to be generally more
  efficient.
```

```
   In the world of ODE methods, Runge-Kutta approaches seem to be more generally favored (ease
  of use, at least as fast, stability) but extrapolation methods have the advantage of being
  variable order and so are suited well for high accuracy needs and long integration times as
  is the case in the present application.  It should be noted that some efficiency is lost in
  using the simpler stepsize control algorithm but the approach is more intuitive.
MODIFICATION HISTORY:
  0210  Scott Williams     First Release (replaced mallocs with VLAs)
*******************************************************************************************/
#define KMAXX  8          // Maximum row number used in the extrapolation.
#define IMAXX  (KMAXX+1)
#define SAFE1  0.25       // Safety factors.
#define SAFE2  0.7
#define REDMAX 1.0e-5     // Maximum factor for stepsize reduction.
#define REDMIN 0.7        // Minimum factor for stepsize reduction.
#define SCALMX 0.1        // 1/SCALMX is maximum factor by which a stepsize can be increased.

void    bdstep(double *y, const double *dydx, const int nv, double *xx, double htry,
            const double eps, const double *yscal, double *hdid, double *hnext,
            void (*derivs)(const double, const double *, double *))
{

    int           i, iq, k, kk, km, reduct, exitflag=0;
    double        eps1, errmax, fact, h, red, scale,work, wrkmin, xest;
    double        x[KMAXX+1], err[KMAXX+1], yerr[nv+1], ysav[nv+1], yseq[nv+1];
    double        d[KMAXX+1][KMAXX+1];
    static int    first=1, kmax, kopt;
    static int    nseq[IMAXX+1] = {0, 2, 4, 6, 8, 10, 12, 14, 16, 18};
    static double epsold = -1.0, xnew;
    static double a[IMAXX+1], alf[KMAXX+1][KMAXX+1];

    if (eps != epsold) {                          // A new tolerance, so reinitialize.
        *hnext = xnew = -1.0e29;                  // "Impossible" values.
        eps1   = SAFE1*eps;
        a[1]   = nseq[1] + 1;                     // Compute work coe.cients Ak.
        for (k=1; k<=KMAXX; k++) a[k+1] = a[k] + nseq[k+1];
        for (iq=2; iq<=KMAXX; iq++) {             // Compute a(k, q).
            for (k=1; k<iq; k++)
                alf[k][iq] = pow(eps1,(a[k+1] - a[iq+1]) / ((a[iq+1] - a[1]+1.0)*(2*k+1)));
        }
        epsold = eps;
        for (kopt=2; kopt<KMAXX; kopt++)          // Determine optimal row # for convergence.
            if (a[kopt+1] > a[kopt]*alf[kopt-1][kopt]) break;
        kmax = kopt;
    }

    h = htry;
    for (i=1; i<=nv; i++)  ysav[i] = y[i];        // Save the starting values.

    // A new stepsize or a new integration: re-establish the order window.
    if (*xx != xnew || h != (*hnext)) {
        first = 1;
        kopt  = kmax;
    }

    reduct=0;
    for (;;) {
        // Evaluate the sequence of modifed midpoint integrations.
        for (k=1;k<=kmax;k++) {
            xnew = (*xx)+h;
            if (xnew == (*xx)) lageos_error("step size underflow in bdstep");

            // step to xx+h using nseq[i] substeps
            mmid(ysav, dydx, nv, *xx, h, nseq[k], yseq, derivs);
            xest = gsl_pow_2(h/nseq[k]);          // Squared, since error series is even.

            // perform extrapolation; result is limit as substep size goes to 0
            rzextr(k, xest, yseq, y, yerr, nv, x, KMAXX+1, d);
            if (k != 1) {                         // Compute normalized error estimate eps(k)
                errmax  = _TINY;
                for (i=1; i<=nv; i++)  errmax = GSL_MAX(errmax, fabs(yerr[i]/yscal[i]));
                errmax /= eps;                    // Scale error relative to tolerance.
                km      = k-1;
                err[km] = pow(errmax/SAFE1, 1.0/(2*km+1));
            }
```

```
            if (k != 1 && (k >= kopt-1 || first)) {// In order window.
                if (errmax < 1.0) {                // Converged.
                    exitflag = 1;
                    break;
                }
                if (k == kmax || k == kopt+1) {    // Check for possible stepsize reduction.
                    red = SAFE2/err[km];
                    break;
                }
                else if (k == kopt && alf[kopt-1][kopt] < err[km]) {
                    red = 1.0/err[km];
                    break;
                }
                else if (kopt == kmax && alf[km][kmax-1] < err[km]) {
                    red = alf[km][kmax-1]*SAFE2/err[km];
                    break;
                }
                else if (alf[km][kopt] < err[km]) {
                    red = alf[km][kopt-1]/err[km];
                    break;
                }
            }
        }
        if (exitflag) break;
        red    = GSL_MIN(red,REDMIN);              // Reduce stepsize by at least REDMIN
        red    = GSL_MAX(red,REDMAX);              // and at most REDMAX.
        h     *= red;
        reduct = 1;
    }                                              // Try again.

    // Successful step taken.
    *xx    = xnew;
    *hdid  = h;
    first  = 0;
    wrkmin =1.0e35;

    // Compute optimal row for convergence and corresponding stepsize.
    for (kk=1; kk<=km; kk++) {
        fact = GSL_MAX(err[kk],SCALMX);
        work = fact*a[kk+1];
        if (work < wrkmin) {
            scale  = fact;
            wrkmin = work;
            kopt   = kk+1;
        }
    }

    *hnext = h/scale;
    // Check for possible order increase, but not if stepsize was just reduced.
    if (kopt >= k && kopt != kmax && !reduct) {
        fact = GSL_MAX(scale/alf[kopt-1][kopt], SCALMX);
        if (a[kopt+1]*fact <= wrkmin) {
            *hnext=h/fact;
            kopt++;
        }
    }
}
#undef KMAXX
#undef IMAXX
#undef SAFE1
#undef SAFE2
#undef REDMAX
#undef REDMIN
#undef SCALMX


/*******************************************************************************************
PROGRAM:  dfridr
  Returns the derivative of a function func at a point x by Ridders' method of polynomial
  extrapolation. The value h is input as an estimated initial stepsize; it need not be small,
  but rather should be an increment in x over which func changes substantially. An estimate
  of the error in the derivative is returned as err.
INPUTS/OUTPUTS/RETURN VALUE:
  func     - pointer to user supplied function to be differentiated
```

```
  x        - value of indep variable at which deriv is required
  h        - initial stepsize to use for the finite differencing
  err      - output error estimate of derivative
  f_status - performance control flag
  rtol     - relative accuracy required of the derivative
  RETURN   - derivative
INCLUDES/EXTERNAL REFERENCES:
  Lincl.h : stdio.h, math.h, gsl_math.h, lageos_warn(),
  Lintgr8.c : NUSE
COMMENTS:
   Adapted from Numerical Recipes in C, 2nd Ed.
MODIFICATION HISTORY:
  0211    Scott Williams   First release
*****************************************************************************************/
#define  CON   1.4                   // Stepsize is decreased by CON at each iteration.
#define  CON2  (CON*CON)
#define  BIG   1.0e30
#define  NTAB  7                     // Sets maximum size of tableau.
#define  SAFE  2.0                   // Return when error is SAFE worse than the best so far.
float   dfridr(float (*func)(float), float x, float h, float *err, FLAG *f_status, float rtol)
{
    int    i, j;
    float  errt, fac, ans, a[NTAB][NTAB];

    if (h == 0.0) lageos_error("h must be nonzero in dfridr.");
    if (fabs(h) <= fabs(x) * GSL_ROOT3_FLT_EPSILON)
        printf("\nWarning [dfridr]: initial stepsize %.4g may be too small to yield "
               "reliable finite difference results\n\n", h);

    // force h to be exactly a machine representable number
    fac = x + h;
    ceil(fac);                       // fcn call forces fac into addressable memory
    h = fac - x;                     // to bypass higher precision internal storage

    a[0][0] = ((*func)(x+h) - (*func)(x-h))/(2.0*h);
    ans     = a[0][0];
    *err    = BIG;
    *f_status = -1;                      /* f_status: -1 = max iteration (NTAB) reached,
                                                       0 = early termination on increase err
                                                       1 = early termination on rtol         */
    for (i=1; i<NTAB; i++) {
        /* Successive columns in the Neville tableau will go to smaller stepsizes and
           higher orders of extrapolation. */
        h       /= CON;
        a[0][i] = ((*func)(x+h) - (*func)(x-h))/(2.0*h);   // Try new, smaller stepsize.
        fac     = CON2;

        // Compute extrapolations of various orders, requiring no new function evaluations.
        for (j=1; j<=i; j++) {
            a[j][i] = (a[j-1][i]*fac - a[j-1][i-1])/(fac - 1.0);
            fac     = CON2*fac;
            errt    = GSL_MAX(fabs(a[j][i] - a[j-1][i]), fabs(a[j][i] - a[j-1][i-1]));

            /* The error strategy is to compare each new extrapolation to one order lower,
               both at the present stepsize and the previous one.  If error is decreased,
               save the improved answer.*/
            if (errt <= *err) {  *err = errt;  ans  = a[j][i];  }
        }
        // if solution is good enough, quit early (added by Scott Williams 0211)
        if (*err <= fabs(rtol*ans))  {  *f_status = 1;  break;  }
        // If higher order is worse by a significant factor SAFE, then quit early.
        if (fabs(a[i][i] - a[i-1][i-1]) >= SAFE*(*err)) { *f_status = 0; break; }
    }
    return ans;
}
#undef  CON
#undef  CON2
#undef  BIG
#undef  NTAB
#undef  SAFE
//*****************************************************************************************
```

Optimization Package

LOPT.C

```
#include    "Lopt.h"
#include    "Lincl.h"
#include    "Lextern.h"

/*****************************************************************************************
PROGRAM:  lageos_optmain
   Control routine for lageos optimization suite - used to determine optimal values of Lageos
  model parameters in correspondence with Pepi Avizonis' empirically obtained spin state data
  for Lageos I.
INPUTS/OUTPUTS/RETURN VALUE:   none
INCLUDES/EXTERNAL REFERENCES:
  Lincl.h   : stdio.h, gsl_math.h, gsl_vector.h,
  - Lprot.h : file_ops(), get_params(), global_alloc(), lageos_error(),
  Lextern.h : gf_out, g_I1, g_I3, g_magscl, g_oblcore, g_rcore, g_save, g_sigcore
  Lopt.h    : NPAR, opt_data_init(), opt_file_ops(), opt_params(), fp_stps, fp_grad,
            : opt_hmax, optv_mp, optv_pmin, opt_scl,
COMMENTS:
MODIFICATION HISTORY:
  0211  Scott Williams     First Release
*****************************************************************************************/
void    lageos_optmain(void)
{

    // Initialize Lageos model & taylor globals for optimization
    global_alloc(1);                                // allocate global matrices & vectors
    get_params(0);                                  // get pre-defined program parameters
    g_save = 0.0;                                   // force no recurrent output
    gf_out = 2;                                     // force no output files
    opt_data_init();                                // get PepiData & set targeted output
    global_alloc(2);                                // allocate internal storage structs
    file_ops(-1);                                   // force no model data output files

    // Open optimization output files and write headers
    opt_file_ops(0);                                // open
    opt_file_ops(1);                                // headers
    opt_file_ops(2);                                // flush

    // Initialize variable model parameters - scale all to internal values of unity
    optv_mp   = gsl_vector_alloc(NPAR);
    optv_pmin = gsl_vector_alloc(NPAR);
    opt_scl[0] = g_rcore;
    opt_scl[1] = g_sigcore;
    opt_scl[2] = g_magscl;
    opt_scl[3] = g_oblcore;
    opt_scl[4] = (g_I3 + g_I1)/2;
    opt_scl[5] = opt_scl[4];
    opt_hmax[4] = 0.5*(g_I3-g_I1)/opt_scl[4];
    opt_hmax[5] = opt_hmax[4];
    gsl_vector_set_all (optv_mp, 1.0);
    gsl_vector_set (optv_mp, 4, g_I1/opt_scl[4]);
    gsl_vector_set (optv_mp, 5, g_I3/opt_scl[5]);
    gsl_vector_memcpy (optv_pmin, optv_mp);

    // Time for the show!
    opt_params();

    // Clean up
    opt_file_ops(3);                                // close optimization output files
    global_alloc(0);                                // free dynamically allocated memory
    file_ops(2);                                    // close data output file streams
}
```

```
/*****************************************************************************************
PROGRAM:  dump_params
   Formatted output of current parameter values
INPUTS/OUTPUTS/RETURN VALUE:   none
INCLUDES/EXTERNAL REFERENCES:
  Lincl.h   : stdio.h, gsl_vector.h, expcat(),
  Lopt.h    : opt_scl
COMMENTS:
MODIFICATION HISTORY:
  0211  Scott Williams     First Release
*****************************************************************************************/
void    dump_params (FILE *fpout, char *subname, const gsl_vector *v_mp, FLAG f_err,
           double *err, FLAG f_endl)
{
    int     expn;
    double  bsn;

    if (fpout == stdout) fprintf(fpout, "\n");
    if (subname != NULL)  fprintf(fpout, "%s", subname);

    // g_rcore
    fprintf(fpout, "%10.4f  |",  gsl_vector_get (v_mp, 0)*opt_scl[0]);
    // g_sigcore
    bsn = expcat(gsl_vector_get (v_mp, 1)*opt_scl[1], &expn);
    fprintf(fpout, "%8.5fe%2d |", bsn, expn);
    // g_magscl
    fprintf(fpout, "%10.4f  |",  gsl_vector_get (v_mp, 2)*opt_scl[2]);
    // g_oblcore
    fprintf(fpout, "%10.5f  |",  gsl_vector_get (v_mp, 3)*opt_scl[3]);
    // g_I1
    bsn = expcat(gsl_vector_get (v_mp, 4)*opt_scl[4], &expn);
    fprintf(fpout, "%9.5fe%d  |", bsn, expn);
    // g_I3
    bsn = expcat(gsl_vector_get (v_mp, 5)*opt_scl[5], &expn);
    fprintf(fpout, "%9.5fe%d  ", bsn, expn);

    if (f_err) {
        bsn = expcat(*err, &expn);
        fprintf(fpout, "||%10.6fe%-+3d", bsn, expn);
    }
    while (f_endl > 0)  {  fprintf(fpout, "\n");  f_endl--;  }
}

/*****************************************************************************************
PROGRAM:  fopt + shells
   Minimization function for Lageos spin model parameter optimization.  Formatted for use with
  gsl multi-dimensional minimization routines.  Computes a net weighted error of model output
  versus Avizonis' spin rate & spin axis orientation data.  The shells are one variable calling
  formats for numerical calculation of the partial derivatives of fopt wrt the variable model
  parameters.
INPUTS/OUTPUTS/RETURN VALUE:
  v_mp - gsl type vector of length NPAR containing the variable model parameters:
         ==>   v_mp = {g_rcore, g_sigcore, g_magscl, g_oblcore, g_I1, g_I3}
  fpar - pointer to minimization function parameters
         ==> pass through fpar a pointer to a double array [3] with elements representing
             the error term weight factors: [0]-spin rate; [1]-righ ascension; [2]-declination
INCLUDES/EXTERNAL REFERENCES:
  Lincl.h : stdio.h, gsl_math.h, gsl_vector.h,
  - Lprot.h : bdstep(), bsstep(), deriv(), deriv_shell_xx(), dump_headers(), dump_log(),
            : get_params(), lageos_spin_xx()
  Lextern.h : fp_log, gf_driver, g_I1, g_I3, g_magscl, g_oblcore, g_rcore, g_sigcore, g_w,
            : optv_mp, opt_scl
  Lopt.h    : COMPTOL, DECAY, HIST, NPAR, opt_scl, optv_mp, optv_w, optv_ra, optv_dec
COMMENTS:
  - The spin rate portion of the error is a scalar -- the difference between best-fit
    exponential decay coefficients of the model and Avizonis' data
  - The components of the spin axis orientation (ra & dec) are treated independently with
    each error a sum-of-squares of the angle differences at the data points; computation
    is done in degrees in part for scaling and in part for easier interpretation of results
  - The three error components are then linearly superposed using the weight factors in the
    fpar array.

  NOTE: The error equation needs to be improved.  For the spatial error, better to compute
```

```
        (at each point) components along the major and minor axes, scaled by the inverse of
        the semi-major axis and semi-minor axis magnitudes respectively.  Also should allow
        the integration start point to vary (add its ra & dec as parameters in the
        optimization)
MODIFICATION HISTORY:
  0211  Scott Williams     First Release
*****************************************************************************************/
float  fopt_shell_float (const float p)
{
    char str[16];
    static int oldndf=-1;

    if (oldndf == opt_ndf) opt_nfopt++;
    else {oldndf = opt_ndf; opt_nfopt = 1; }

    gsl_vector_set(optv_mp, opt_ndf, (double) p);
    sprintf (str, "fopt_shell_%d", opt_ndf);
    printf ("%-14s|\n", str);
    return  (float) fopt(optv_mp, (void *) opt_wts);
}
double  fopt_shell (const double p, void * fpar)
{
    char str[16];
    gsl_vector_set(optv_mp, opt_ndf, p);
    sprintf (str, "fopt_shell_%d", opt_ndf);
    printf ("%-14s|\n", str);
    return  fopt(optv_mp, fpar);
}
// *****************************************************************************************
double  fopt (const gsl_vector * v_mp, void * fpar)
{
    char           str[70];
    int            i;
    double         c0, cov00, cov01, cov11, chisq, decay;
    double         logw[opt_n], rate_err=0., ra_err=0., dec_err=0.;
    double         *errwt = (double *) fpar;
    static int     indx, count;
    static double  oldpar[HIST][NPAR], result[HIST], errmin0, errmin[4]={0.0, 0.0, 0.0, 1e30};


// ~~~~~~~~~~~~~~~~~~~~ Set variable params & check for redundent computation ~~~~~~~~~~~~~~~~~~
    // set new model parameter values
    g_rcore   = opt_scl[0]*gsl_vector_get (v_mp, 0);
    g_sigcore = opt_scl[1]*gsl_vector_get (v_mp, 1);
    g_magscl  = opt_scl[2]*gsl_vector_get (v_mp, 2);
    g_oblcore = opt_scl[3]*gsl_vector_get (v_mp, 3);
    g_I1      = opt_scl[4]*gsl_vector_get (v_mp, 4);
    g_I3      = opt_scl[5]*gsl_vector_get (v_mp, 5);

    // show where we are
    printf("%-14s|\n\n","fopt (in)");
    sprintf(str,"%-14s|","fopt (out)");

    // check if already have the value in recent history (saves lengthy integration time)
    for (i=0; i < HIST; i++) {
        FLAG f_tst=1;
        int  j;
        for (j=0; j<NPAR; j++) {
           // internal values scaled to order unity so this is actually a relative comparison
            if (fabs(gsl_vector_get(v_mp,j)-oldpar[i][j])>COMPTOL) { f_tst = 0;  break;  }
        }
        if (f_tst) {
            dump_params(stdout, str, v_mp, 1, &result[i], 1);
            if (count > HIST) {
                sprintf (str, "Multi-dimensional minimization unable to make further progress");
                fprintf(fp_stps, "%s", str);
                lageos_error(str);
            }
            count += 1;
            return result[i];
        }
    }
    count = 0;
// ~~~~~~~~~~~~~~~~~~~~~~~~ Proceed with integration to get new data set ~~~~~~~~~~~~~~~~~~~~~~~
```

```
    get_params (1);                                 // recompute derived parameters
    dump_headers();
    switch (gf_driver) {
      case 1:  lageos_spin_nr ((void *) deriv, (void *) bdstep); break;
      case 2:  lageos_spin_rk ((void *) deriv_shell_rk); break;
      case 3:  lageos_spin_de (deriv_shell_de); break;
      default: lageos_spin_nr ((void *) deriv, (void *) bsstep);
    }
    banner(stdout, 1, '=', 0, 0, 105, 0, 0, 0, 0);

// ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ Compute individual error terms ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
    for (i=0; i<opt_n; i++) {
        logw[i]   = log(g_w[i].mag);
        ra_err   += gsl_pow_2 (M_DPR*(gsl_vector_get (optv_ra, i)  - g_w[i].ra));
        dec_err  += gsl_pow_2 (M_DPR*(gsl_vector_get (optv_dec, i) - g_w[i].dec));
    }
    gsl_fit_linear (opt_t, 1, logw, 1, opt_n, &c0, &decay, &cov00, &cov01, &cov11, &chisq);
    rate_err     = opt_n*gsl_pow_2(decay - DECAY);
    rate_err    *= errwt[0];
    ra_err      *= errwt[1];
    dec_err     *= errwt[2];

    // store results for comparison on input
    indx++;
    indx = (indx==HIST) ? 0 : indx;
    result[indx]    = gsl_pow_2 (rate_err + ra_err + dec_err);
    //result[indx]    = rate_err + ra_err + dec_err;
    oldpar[indx][0] = g_rcore/opt_scl[0];
    oldpar[indx][1] = g_sigcore/opt_scl[1];
    oldpar[indx][2] = g_magscl/opt_scl[2];
    oldpar[indx][3] = g_oblcore/opt_scl[3];
    oldpar[indx][4] = g_I1/opt_scl[4];
    oldpar[indx][5] = g_I3/opt_scl[5];
    // save current set if better than any before
    if (result[indx] < errmin[3]) {
        errmin[0] = rate_err;
        errmin[1] = ra_err;
        errmin[2] = dec_err;
        errmin[3] = result[indx];
        for (i=0; i<NPAR; i++) gsl_vector_set (optv_pmin, i, oldpar[indx][i]);
    }

    // print results
    dump_params(stdout, str, v_mp, 1, &result[indx], 1);
    if (optf_out) {
        int    ert;
        sprintf(str, "%3d ||    ||", opt_nit);
        dump_params(fp_fvals, str, v_mp, 0, NULL, 0);
        rate_err = expcat (rate_err, &ert);
        fprintf(fp_fvals,"||%9.5fe%-+3d |%#13.6g |%#13.6g ||%#13.7g\n",
                         rate_err, ert, ra_err, dec_err, result[indx]);
        if (errmin[3] < result[indx] && errmin[3] != errmin0) {
            dump_params(fp_fvals, "   BEST   ||", optv_pmin, 0, NULL, 0);
            rate_err = expcat (errmin[0], &ert);
            fprintf(fp_fvals,"||%9.5fe%-+3d |%#13.6g |%#13.6g ||%#13.7g\n", rate_err, ert,
                             errmin[1], errmin[2], errmin[3]);
            errmin0 = errmin[3];
        }
        fflush(fp_fvals);
    }
    return result[indx];
}

/*****************************************************************************************
PROGRAM:  dfopt, fdfopt
   Computes the gradient of fopt with respect to the variable model parameters using
  numerical differentiation (centerd difference); fdfopt combines the computations of fopt
  and dfopt into a single routine (used by the minimization routines).
INPUTS/OUTPUTS/RETURN VALUE:
  v_mp    - gsl type vector of length NPAR containing the variable model parameters:
         ==>   v_mp = {g_rcore, g_sigcore, g_magscl, g_oblcore, g_I1, g_I3}
  fpar    - pointer to minimization function parameters (see fopt description)
  v_fgrad - resulting gsl type vector of length 2 containing the gradient of fopt, i.e., the
```

```
            partial derivs of fopt wrt the variable model parameters
INCLUDES/EXTERNAL REFERENCES:
  Lincl.h : stdio.h, gsl_diff.h, gsl_math.h, gsl_vector.h
  - Lprot.h : elapsed_time(), dump_params(), opt_file_ops(),
  Lopt.h  : NPAR, fp_fvals, fp_grad, opt_ndf, optv_mp,
COMMENTS:
   Initially built this routine to use gsl's numerical differentiation formula.  However, that
  routine is only available in double precision and allows minimal user control.  This makes
  it unreliable to use with a function based on numerical differentiation since the results
  of the integration are most certainly less than full double precision.
   So, this routine now utilizes a numerical differentiation technique adapted from Numerical
  Recipes in C based on the idea of polynomial extrapolation.  The routine is float precision,
  though it could just as easily be double b/c the user can specify a desired tolerance and
  evaluations are not dependent on extremely small relative stepsizes.
MODIFICATION HISTORY:
  0211  Scott Williams     First Release
*******************************************************************************************/
void    dfopt (const gsl_vector * v_mp, void * fpar, gsl_vector * v_fgrad)
{
    char          str[16];
    FLAG          f_status;
    int           i;
    long          emin;
    float         p[NPAR], dfdmp[NPAR], err[NPAR];                      // nr_c: dfridr
    double        esec, nrmdf=0, nrmerr=0;
    static int    count;
    static float  h[NPAR];                                              // nr_c: dfridr

    //double      p[NPAR], dfdmp[NPAR], err[NPAR];                        // gsl_diff_central
    //gsl_function  F;
    //F.function = &fopt_shell;
    //F.params   = fpar;
    //for (i=0; i< NPAR; i++) p[i] = gsl_vector_get (v_mp, i);            // gsl_diff_central

    for (i=0; i< NPAR; i++) p[i] = (float) gsl_vector_get (v_mp, i);      // nr_c: dfridr

    printf("%-14s|\n","dfopt (in)");
    fprintf(fp_fvals,"\n%3d ||%3d || param ", opt_nit, count);
    fprintf(fp_grad, "%3d ||%3d ", opt_nit, count);

    // get derivative of model parameters
    optf_out = 0;                                   // suppress outputs during deriv calcs
    for (opt_ndf = 0; opt_ndf<NPAR; opt_ndf++) {
        int     k=opt_ndf;                          // just begin lazy - opt_ndf cumbersome

        elapsed_time(&emin, &esec);
        fprintf(fp_fvals," [%d]{%03ld:%04.1f, ", k, emin, esec);
        fflush(fp_fvals);
        gsl_vector_memcpy (optv_mp, v_mp);

        // Begin nc_c: dfridr
        if (!h[k]) h[k] = opt_hmax[k];
        // Make sure h not too big or too small
        h[k] = GSL_MIN (GSL_MAX(h[k], MINHF*opt_hmax[k]), opt_hmax[k]);
        opt_nfopt = 0;

        // limit number of iterations; will use best of all results obtained if no convergence
        for (i=0; i<4; i++) {
            FLAG    f_stop=0;
            int     j;
            double  fac, olderr=-999, olddf, oldfac;

            // check to see if parameter excluded
            for (j=0; j<NPARX; j++)  if (k == opt_nx[j]) { f_stop = 1; opt_nfopt = 0; break; }
            if (f_stop) {  dfdmp[k] = 0.0; err[k] = 0.0;  break;  }

            // Get the derivative
            dfdmp[k] = (double) dfridr(fopt_shell_float, p[k], h[k], &err[k], &f_status, DTOL);

            // done & don't need any adjustments if error < DTOL
            if (f_status == 1) break;
            // otherwise print result indicator and adjust stepsize seed for next go-round
            if (!f_status) {  fprintf(fp_fvals, "$%d.", opt_nfopt);  fac = SHRSTP; }
            else           {  fprintf(fp_fvals, "*%d.", opt_nfopt);  fac = GROSTP;  }
```

```
            h[k] *= fac;

            // accept looser but adequate step to avoid too many fopt calls
            if (err[k] <= DRTOL*fabs(dfdmp[k]))  break;

            // save old results before repeat
            if (olderr == -999) {
                oldfac = fac;
                olderr = err[k];
                olddf  = dfdmp[k];
                continue;
            }

            // if get to here then multiple times through & df still rejected
            if (err[k] <= olderr) {                 // result improved with previous scaling of
                olderr = err[k];                    // h so make sure to go same way
                olddf  = dfdmp[k];
                if (fac != oldfac)  h[k] *= (oldfac/fac);  // undo current step and apply prev
            }
            else {                                  // went the wrong way so restore prev
                static int repeat = 0;
                err[k]   = olderr;                  // results & try other direction
                dfdmp[k] = olddf;
                if (repeat) break;                  // if here a 2nd time then just oscillating
                h[k] /= (oldfac*fac);               // reset h
                oldfac = (oldfac < 1) ? GROSTP : SHRSTP;
                h[k] *= oldfac;
                repeat = 1;
            }
        }
        fprintf(fp_fvals,"%-2d}; ", opt_nfopt);
        // End nc_c: dfridr

/*      // Begin gsl_diff_central
        gsl_diff_central (&F, p[k], &dfdmp[k], &err[k]);
        gsl_vector_set (v_fgrad, k, dfdmp[k]);
*/
        gsl_vector_set (v_fgrad, k, (double) dfdmp[k]);
        fprintf(fp_grad, "||%#11.4g |%#11.4g ", dfdmp[k], err[k]);
        fflush(fp_grad);
        nrmdf  += gsl_pow_2(dfdmp[k]);
        nrmerr += gsl_pow_2(err[k]);
    }

    // Print status to screen
    sprintf(str, "%-14s|","dfopt (out)");
    dump_params(stdout, str, v_mp, 0, NULL, 1);
    printf("%s", str);
    for (i=0; i<NPAR; i++)   printf(" d%d = %.4g |", i, dfdmp[i]);
    printf("\n");  fprintf(fp_fvals, "\n");
    fprintf(fp_grad,"||%#11.4g |%#11.4g \n", sqrt(nrmdf), sqrt(nrmerr));


    opt_file_ops(2);                                // flush buffers
    count++;
    optf_out = 1;
}
// Computes fopt and dfopt simultanesouly
void    fdfopt (const gsl_vector * v_mp, void * fpar, double *fcn, gsl_vector * v_fgrad)
{
    printf("%-14s|\n","fdfopt (in)");

    *fcn = fopt (v_mp, fpar);
    dfopt (v_mp, fpar, v_fgrad);

    printf("%-14s|\n","fdfopt (out)");
}

/*****************************************************************************************
PROGRAM:  opt_data_init
   Retrieves PepiData within the interval g_start to g_stop (ascending or descending) and
  stores the data in vectors used by the fopt functions.  Resets g_stop if need be to the
  last element of the set.  Also reconstructs g_out (and its friends g_nout, g_outndx) to
  be in correspondence with the PepiData elements.
```

```
INPUTS/OUTPUTS/RETURN VALUE:   none
INCLUDES/EXTERNAL REFERENCES:
  Lincl.h   : math.h, gsl_math.h, gsl_vector.h,
  - Lprot.h : lageos_error()
  Lextern.h : gf_idir, g_nout, g_out, g_outndx, g_start, g_stop,
  Lopt.h    : NDATA, PepiData, opt_n, optv_w, optv_ra, optv_dec
COMMENTS:
MODIFICATION HISTORY:
  0211  Scott Williams     First Release
*****************************************************************************************/
void    opt_data_init (void)
{
    int    i, lndx, undx;

    // hunt for & count PepiData elements between start & stop
    lndx = 0;
    undx = NDATA-1;
    while (PepiData[lndx][0] < GSL_MIN(g_start, g_stop)) lndx++;
    while (PepiData[undx][0] > GSL_MAX(g_start, g_stop)) undx--;
    if ((opt_n = undx-lndx+1) < 1)
        lageos_error ("No data elements within optimization integration interval");
    if ((g_start != PepiData[lndx][0]) && (g_start != PepiData[undx][0]))
        lageos_error ("Invalid initial state for optimization--no correspondence w/data set");
    // store valid PepiData elements in working arrays & reinitialize targeted output array
    gsl_vector_free (g_out);
    g_out    = gsl_vector_alloc(opt_n);
    optv_w   = gsl_vector_alloc(opt_n);
    optv_ra  = gsl_vector_alloc(opt_n);
    optv_dec = gsl_vector_alloc(opt_n);

    if (gf_idir) {                                  // integration in positive time direction
        // force integration stop to last element in set
        g_stop  = PepiData[undx][0];
        for (i=0; i<opt_n; i++) {
            gsl_vector_set (g_out,    i, PepiData[lndx+i][0]);
            gsl_vector_set (optv_w,   i, PepiData[lndx+i][1]);
            gsl_vector_set (optv_ra,  i, PepiData[lndx+i][2]);
            gsl_vector_set (optv_dec, i, PepiData[lndx+i][3]);
        }
    } else {                                        // integration in negative time direction
        // force integration stop to last element in set
        g_stop  = PepiData[lndx][0];
        for (i=0; i<opt_n; i++) {
            gsl_vector_set (g_out,    i, PepiData[undx-i][0]);
            gsl_vector_set (optv_w,   i, PepiData[undx-i][1]);
            gsl_vector_set (optv_ra,  i, PepiData[undx-i][2]);
            gsl_vector_set (optv_dec, i, PepiData[undx-i][3]);
        }
    }
    opt_t    = gsl_vector_ptr (g_out, 0);
    g_outndx = 1;                                   // g_outndx pts to first element past start
    g_nout   = opt_n;                               // g_nout is number of targeted outputs
}

/*****************************************************************************************
PROGRAM:  opt_file_ops
  Performs output file maintenance for optimization routine (open, close, flush, headers)
INPUTS/OUTPUTS/RETURN VALUE:
  f_mode - mode flag: 0 = open files for standard output,
                      1 = write headers
                      2 = flush output streams,
                      3 = close files
                     -1 = Turn on standard lageos output files
                     -2 = turn off standard lageos output files
INCLUDES/EXTERNAL REFERENCES:
  Lincl.h   : stdio.h, FLAG
  Lextern.h : fp_euler, fp_angvel, fp_angmom, fp_log, fp_orbit
*****************************************************************************************/
void    opt_file_ops(const FLAG f_mode)
{
    switch (f_mode)
    {
        case -2:    {   // return to 'no output state'
            file_ops(1);                            // flush the buffers
```

```
            file_ops(2);                            // close the files
            file_ops(-1);                           // reset to null streams
            break; }
        case -1:    {   // re-initialize standard Lageos output files for output
            char filename[20];
            file_ops(2);                           // clean up first just in case!
            // Euler output file
            sprintf(filename,"l_euler_%02d.txt", opt_nit);
            if ((fp_euler  = fopen(filename,"w")) == NULL)
                lageos_error("Could not open euler angle output file");

            // Angular Velocity output file
            sprintf(filename,"l_angvel_%02d.txt", opt_nit);
            if ((fp_angvel = fopen(filename,"w")) == NULL)
                lageos_error("Could not open angular velocity output file");

            // Angular Momentum output file
            sprintf(filename,"l_angmom_%02d.txt", opt_nit);
            if ((fp_angmom = fopen(filename,"w")) == NULL)
                lageos_error("Could not open angular momentum output file");

            // Log output file
            sprintf(filename,"l_log_%02d.txt", opt_nit);
            if ((fp_log    = fopen(filename,"w")) == NULL)
                lageos_error("Could not open program log output file");

            // Orbit parameters output file
            sprintf(filename,"l_orbit_%02d.txt", opt_nit);
            if ((fp_orbit  = fopen(filename,"w")) == NULL)
                lageos_error("Could not open orbit output file");
            break; }
        case 0:    {    // open files for output
            if ((fp_stps = fopen("l_optstps.txt", "w")) == NULL)
                lageos_error("Could not open optimization iteration output file");
            if ((fp_grad = fopen("l_optgrad.txt", "w")) == NULL)
                lageos_error("Could not open optimization iteration output file");
            if ((fp_fvals = fopen("l_optfvals.txt", "w")) == NULL)
                lageos_error("Could not open optimization function output file");
            break; }
        case 1:    {    // write header lines
            dump_log(fp_stps, 0, NULL, NULL, NULL, NULL, NULL, NULL, NULL);
            banner(fp_stps, 1, '=', 0, 0, 105, 0, 0, 0, 0);
            fprintf(fp_stps," it# ||   g_rcore  | g_sigcore  |  g_magscl  |  g_oblcore |"
                            "    g_I1     |    g_I3     ||    net_err    ||  |dparams| \n");
            banner(fp_stps, 1, '=', 0, 0, 116, 0, 0, 0, 0);
            fprintf(fp_fvals,"it# || g# ||   g_rcore  | g_sigcore  |  g_magscl  |  g_oblcore |"
                             "    g_I1     |    g_I3     ||   rate_err   |    ra_err    |"
                             "    dec_err   ||  total_error\n");
            banner(fp_fvals, 1, '=', 0, 0, 154, 0, 0, 0, 0);
            fprintf(fp_grad,"it# || g# ||   drcore   |   drc_err  ||  dsigcore  |   dsc_err  |"
                            "|   dmagdir  |   dmd_err  ||  doblcore  |   doc_err  |"
                            "|     dI1    |   dI1_err  ||     dI3    |   dI3_err  |"
                            "|   |grad|   |   RSS_err\n");
            banner(fp_grad, 1, '=', 0, 0, 199, 0, 0, 0, 0);
            break; }
        case 2:    {    // flush buffers
            fflush(fp_stps);
            fflush(fp_fvals);
            fflush(fp_grad);
            break; }
        case 3:    {    // close output files
            fclose(fp_stps);
            fclose(fp_grad);
            fclose(fp_fvals);
            break; }
    }
}

/*****************************************************************************************
PROGRAM:  opt_params
   Driver routine for the multi-dimensional minimization.
INPUTS/OUTPUTS/RETURN VALUE:   none
INCLUDES/EXTERNAL REFERENCES:
  Lincl.h   : stdio.h, gsl_math.h, gsl_multimin.h, gsl_vector.h,
```

```
  - Lprot.h : lageos_error(),
  Lopt.h    : GRADTOL, NPAR, SETTOL, STEP1, fopt(), dfopt(), fdfopt(), opt_data_init(),
            : opt_params(), fp_stps, optv_mp, opt_wts
COMMENTS:
MODIFICATION HISTORY:
  0211  Scott Williams     First Release
*******************************************************************************************/
void    opt_params(void)
{
         //size_t opt_nit=0;
    char   str[30];
    int    status = GSL_CONTINUE, edpm=0;
    long   emin;
    double esec, dparmag=0, bdpm=0;
    gsl_vector  * v_mpar, * v_mpold;
    gsl_multimin_function_fdf             fmp;
    gsl_multimin_fdfminimizer            *s;
    const gsl_multimin_fdfminimizer_type *T;

    // Initialize the multimin function data type
    fmp.f   = &fopt;
    fmp.df  = &dfopt;
    fmp.fdf = &fdfopt;
    fmp.n   = NPAR;
    fmp.params = (void *) opt_wts;

    // Initialize variable model parameters vector
    v_mpold = gsl_vector_calloc(NPAR);
    v_mpar  = gsl_vector_alloc(NPAR);
    gsl_vector_memcpy (v_mpar, optv_mp);

    // Identify multi-dimensional minimizer method & set pointer to it
    switch (MINMETH) {
      case 1:  T = gsl_multimin_fdfminimizer_conjugate_fr;  break;
      case 2:  T = gsl_multimin_fdfminimizer_conjugate_pr;  break;
      default: T = gsl_multimin_fdfminimizer_vector_bfgs;
    }
    s = gsl_multimin_fdfminimizer_alloc (T, fmp.n);

    // Initialize minimizer
    printf("%-14s| initializing fdfminimizer\n\n","opt_params");
    gsl_multimin_fdfminimizer_set (s, &fmp, v_mpar, STEP1, SETTOL);


    // find the minimum!
    while (status == GSL_CONTINUE && opt_nit < 100) {
        // output current data point
        sprintf(str, "%-14s|%2d |", "opt_params", opt_nit);
        dump_params(stdout, str, s->x, 1, &(s->f), 1);
        sprintf(str,"%4d ||", opt_nit);
        dump_params(fp_stps, str, s->x, 1, &(s->f), 0);
        fprintf(fp_stps," ||%8.4fe%-+3d\n", bdpm, edpm);
        fflush(fp_stps);

        // take a step
        opt_nit++;
        if ((status = gsl_multimin_fdfminimizer_iterate (s)))
            lageos_error("Problem encountered with multi-dimensional optimization");

        // check convergence of gradient
        status = gsl_multimin_test_gradient (s->gradient, GRADTOL);

        // check for progress with parameter values
        gsl_vector_sub (v_mpold, s->x);
        dparmag = gsl_blas_dnrm2(v_mpold);
        bdpm = expcat(dparmag, &edpm);
        if(dparmag < PARTOL)  {
            status = GSL_SUCCESS;
            fprintf(fp_stps, "Incremental change in parameters <= PARTOL; Final Optimzation "
                             "step is:\n");
        }
        gsl_vector_memcpy (v_mpold, s->x);
    }
```

```
    // output final result
    sprintf(str, "%-14s|it#=%2d |", "opt_params", opt_nit);
    dump_params(stdout, str, s->x, 1, &(s->f), 1);
    sprintf(str,"%4d ||", opt_nit);
    dump_params(fp_stps, str, s->x, 1, &(s->f), 0);
    fprintf(fp_stps," ||%8.4fe%-+3d\n", bdpm, edpm);
    elapsed_time(&emin, &esec);
    fprintf(fp_fvals, "End optimization at %3ld:%04.1f.\n", emin, esec);
    opt_file_ops (2);

    gsl_multimin_fdfminimizer_free (s);
    gsl_vector_free (v_mpar);
}
//*************************************************************************************************
```

LOPT.H

```
#include    <gsl/gsl_diff.h>        // Finite differencing (derivatives) routines
#include    <gsl/gsl_fit.h>         // Linear Least Squares data fitting routines
#include    <gsl/gsl_multimin.h>    // Multi-dimensional minimization routines
#include    <g2c.h>

/* each row should be the data corresponding to the time in the first column; time column
   !!! must !!! be stored in ascending order                                              */
#define  NDATA 29
const double  PepiData[NDATA][4] =
            {  /*     JD2K        w (rad/s)       ra (rad)    dec (rad)   */
                { -4110.109722, 1.67061561E-01,  2.65063154, -1.27269409 },
                { -4040.265278, 1.57631342E-01, -0.95661496, -1.44303823 },
                { -3783.213194, 1.25362835E-01, -0.26179939, -1.36135682 },
                { -2826.333206, 5.36932602E-02, -1.50866261, -1.32121424 },
                { -2772.093322, 5.11160536E-02, -1.03253679, -1.35455003 },
                { -2769.118056, 5.13206347E-02, -0.96377081, -1.34146006 },
                { -2761.291470, 5.07117458E-02, -0.91559973, -1.33395515 },
                { -2758.314155, 5.12745659E-02, -0.89081605, -1.34425259 },
                { -2712.420775, 4.88697621E-02, -0.79726640, -1.40673538 },
                { -2678.446690, 4.71003396E-02,  0.00628319, -1.40795711 },
                { -2650.489155, 4.66804258E-02,  0.35901423, -1.40481551 },
                { -2647.443750, 4.71781447E-02,  0.42778020, -1.39329634 },
                { -2642.493750, 4.66804258E-02,  0.48781953, -1.38544236 },
                { -2633.503472, 4.55964101E-02,  0.52569317, -1.39713607 },
                { -2625.526389, 4.47201801E-02,  0.67753682, -1.37776291 },
                { -2443.370139, 3.82188887E-02,  1.53344628, -1.28875112 },
                { -2439.306250, 3.76689767E-02,  1.55299397, -1.27234502 },
                { -2430.370139, 3.81030037E-02,  1.63903870, -1.24372162 },
                { -2404.369444, 3.72800837E-02,  1.56974913, -1.28351513 },
                { -2359.197917, 3.58424718E-02,  1.63153378, -1.28054807 },
                { -2299.422222, 3.38533691E-02,  1.75964095, -1.26030225 },
                { -2268.318750, 3.32092247E-02,  1.88617732, -1.30882241 },
                { -2241.531944, 3.21884493E-02,  2.00451065, -1.33465328 },
                { -2240.446528, 3.15262685E-02,  2.02353473, -1.35786616 },
                { -1771.164583, 2.01384144E-02, -1.02468280, -1.15523343 },
                { -1755.327083, 2.12269774E-02, -0.03385939, -1.16169115 },
                { -1732.318056, 2.10845145E-02,  0.65083328, -1.21876342 },
                { -1284.500000, 1.28228272E-02,  1.32557757, -1.17914444 },
                { -1172.500000, 1.28228272E-02,  1.20864051, -1.24738682 } };

#define   DECAY    -8.95e-4 //-8.888779e-4 // exponential decay coefficient of spin rate
#define   NPAR     6                       /* # of variable model parameters in optimziation
                                          {g_rcore, g_sigcore, g_magscl, g_oblcore, g_I1, g_I3} */
#define   NPARX    2                       // # of NPAR to exclude (see opt_nx)
#define   HIST     10*NPAR                 // number of historical results to store in fopt
                                           // to prevent redundent computation
#define   MINMETH  2                       // method: 1=conj_fr, 2=conj_pr, 3=bfgs
#define   COMPTOL  1.0e-14                 // relative precision with which to determine
                                           // "equality" with historical input values
#define   SETTOL   5.0e-3                  // Tolerance for fdfminimizer initialization
#define   GRADTOL  1.0e-2                  // Tolerance for gradient convergence
#define   PARTOL   1.0e-5                  // Tolerance for parameter convergence
#define   DTOL     1.0e-3                  // Relative accuracy desired in derivative calc
```

```
#define   DRTOL    0.25                    // Relative accuracy REQUIRED in derivative calcs
#define   STEP1    0.01                    // Initial stepsize to try along line minimization
#define   GROSTP   1.5                     // Stepsize adjustment factors for dfopt
#define   SHRSTP   0.35                    // routine
#define   MINHF    1e-2                    // minimum factor of opt_hmax values allowed

FILE        *fp_stps;                      // Multi-dimensional minimization iterates file
FILE        *fp_grad;                      // Numerical derivatives file
FILE        *fp_fvals;                     // fopt function output file
int          optf_out=1;                   // trigger std output file generation
int          opt_n;                        // # of data elements in integration interval
int          opt_ndf;                      // index of parameter to vary in differentiation
int          opt_nx[NPARX] = {1,3};        /* indices of parameters to exclude from optimizatn
                                              (i.e., hold const); forces partials to 0; set
                                              NPARX=1 and opt_nx[0]>=NPAR for no exclusions  */
int          opt_nfopt=1;                  // number of fopt calls for each partial deriv
size_t       opt_nit;                      // minimization iteration number
double       opt_wts[3] = {1e12, 1.0, 1.0};
                                           // error weights with scaling: {decay, ra, dec}
double       opt_scl[NPAR];                /* scaling factors for variable model parameters -
                                              use to scale internal values to order unity;
                                         {g_rcore, g_sigcore, g_magscl, g_oblcore, g_I1, g_I3} */
float        opt_hmax[NPAR] = {5e-3, 0.01, 0.1, 0.25, 0.025, 0.025};
                                           /* stepsize seeds for nr_c dfridr relative to the
                                              internal scaling of ops_scl
                                         {g_rcore, g_sigcore, g_magscl, g_oblcore, g_I1, g_I3} */
double      *opt_t;                        // pointer to JD2K optv_t of PepiData elements
gsl_vector  *optv_mp;                      // vector of variable model parameters
gsl_vector  *optv_pmin;                    // vector of params for best internal fopt value
gsl_vector  *optv_w;                       //  / vectors of PepiData elements in integration
gsl_vector  *optv_ra;                      // <  interval; sorted in correspondence w/JD2K
gsl_vector  *optv_dec;                     //  \ values (ascend/descend depends on gf_idir)
//*************************************************************************************************
```